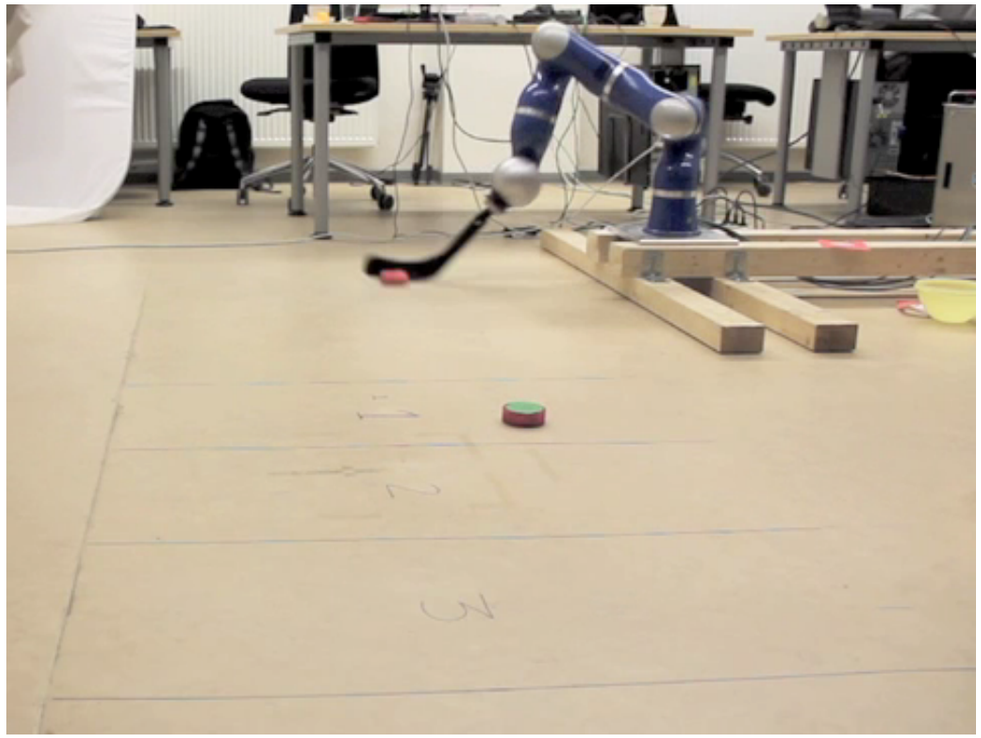
Robot Autonomy

Lecture 2:
MDPs and Reinforcement Learning

Oliver Kroemer

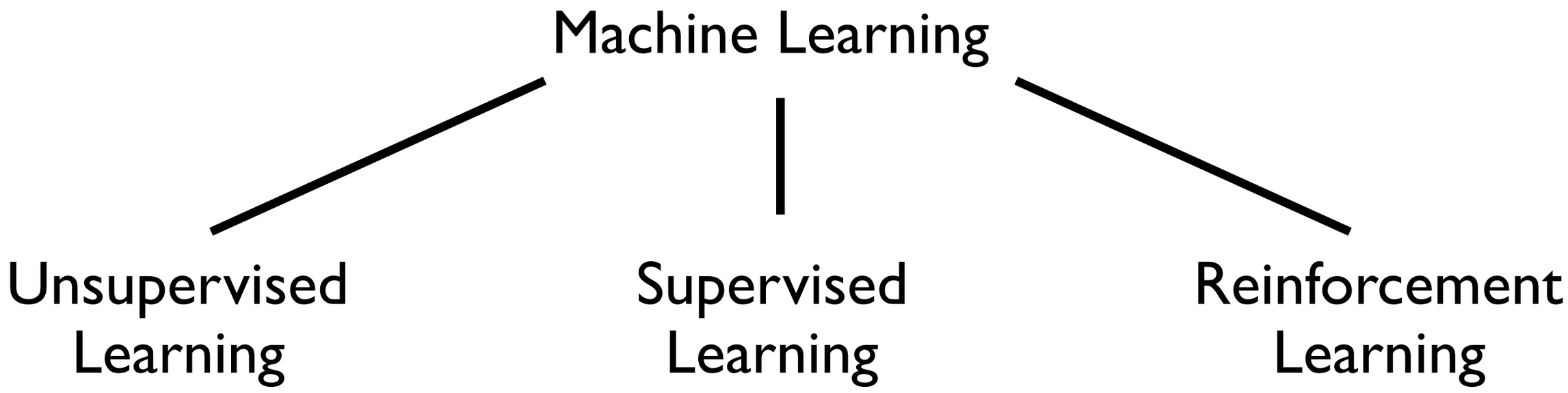
Motivation

- Want robots to perform complex adaptive behaviors

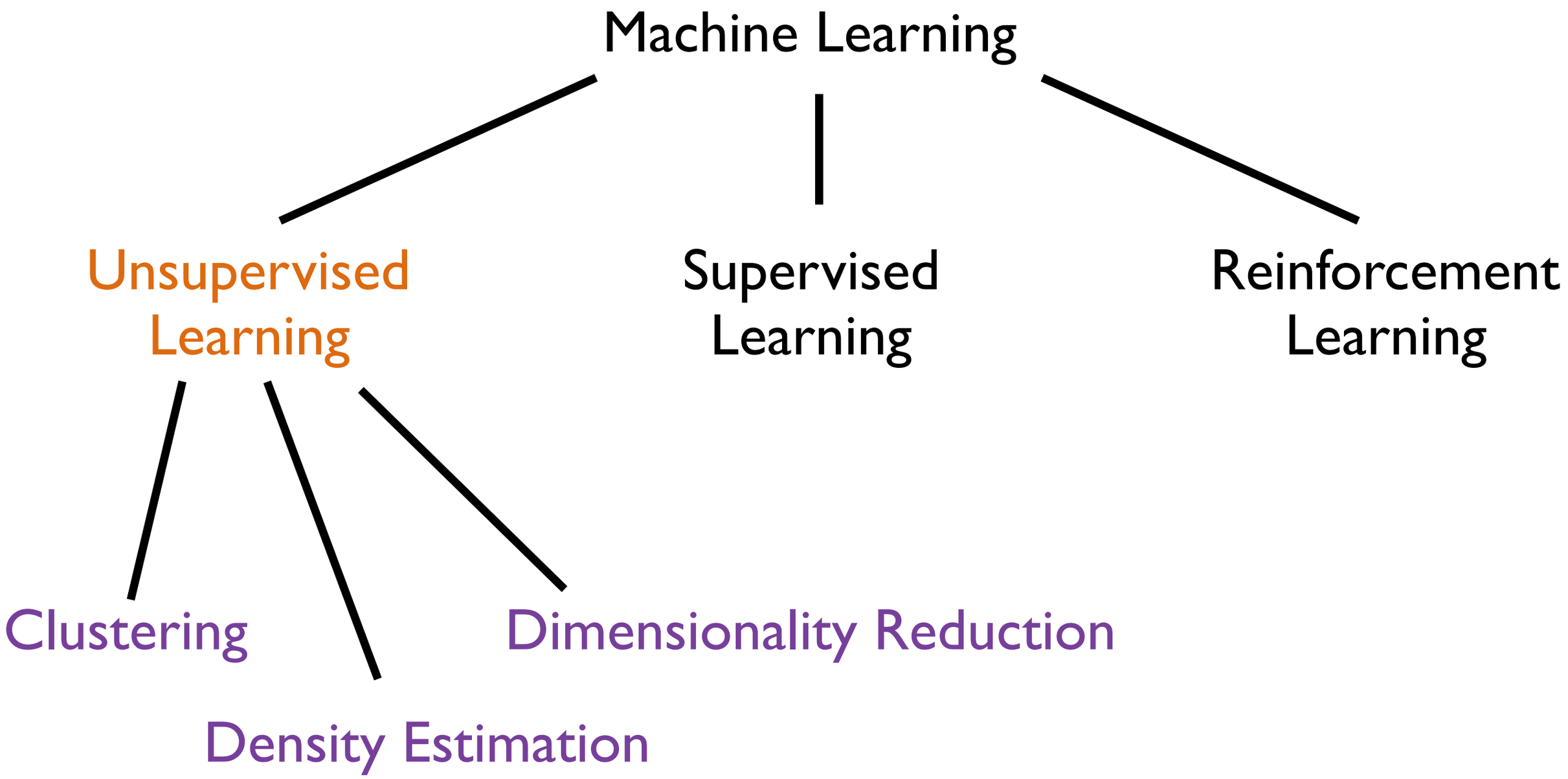


- Programming these behaviors by hand is not trivial
- Want to adapt to complex unstructured environments
- Often easier to have robot learn from experience

Machine Learning

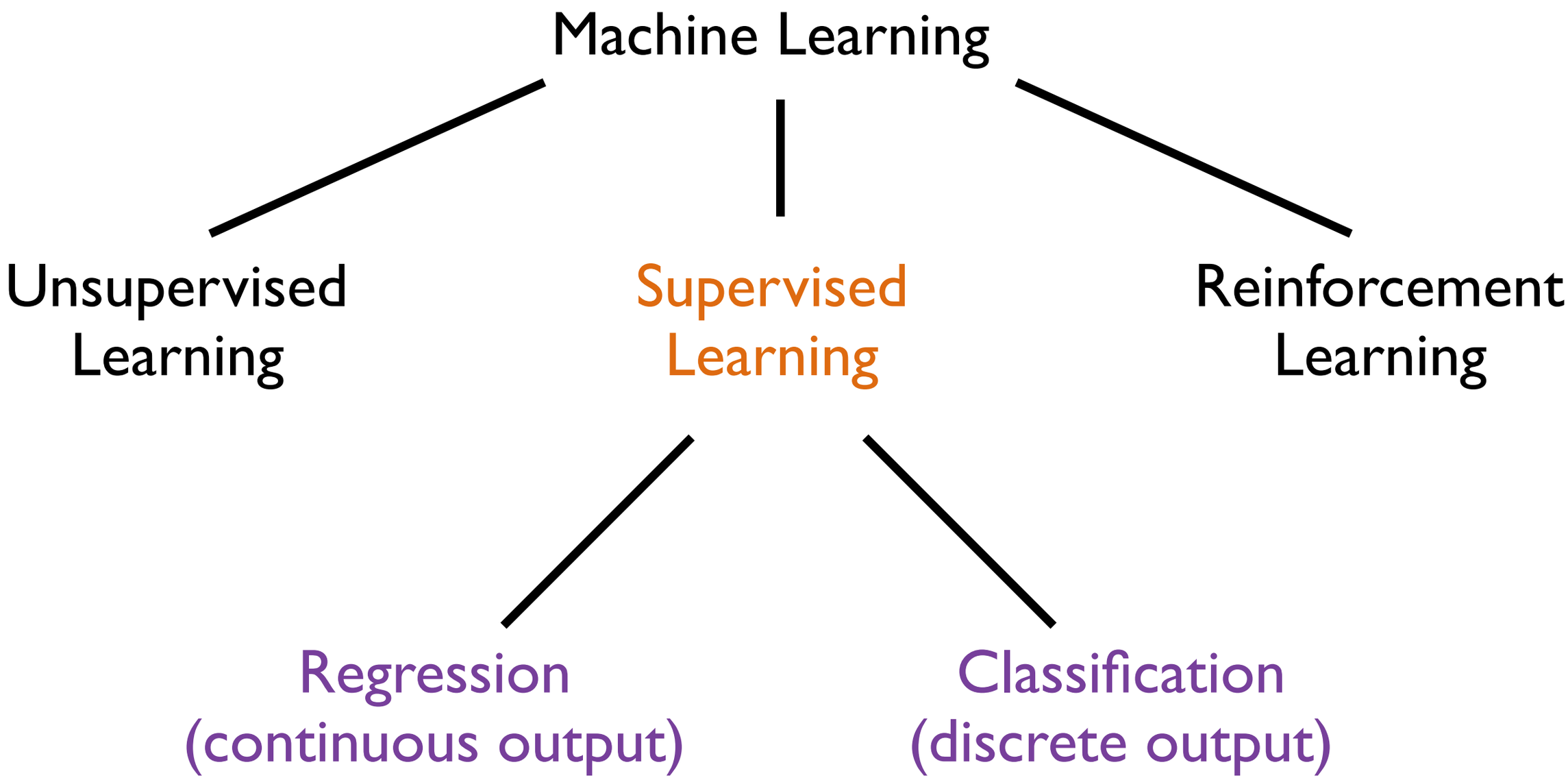


Machine Learning



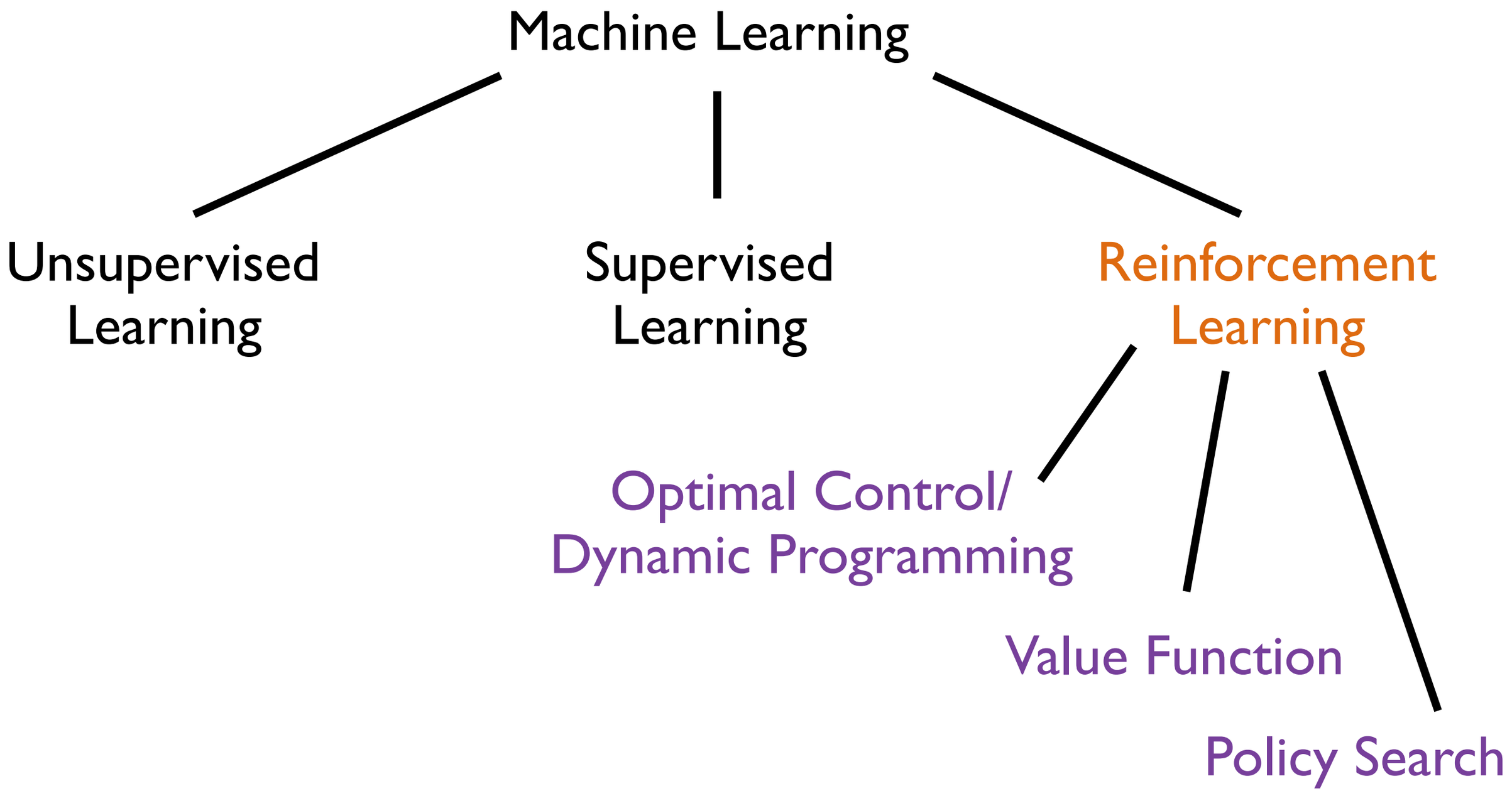
learn structure from input information only

Machine Learning



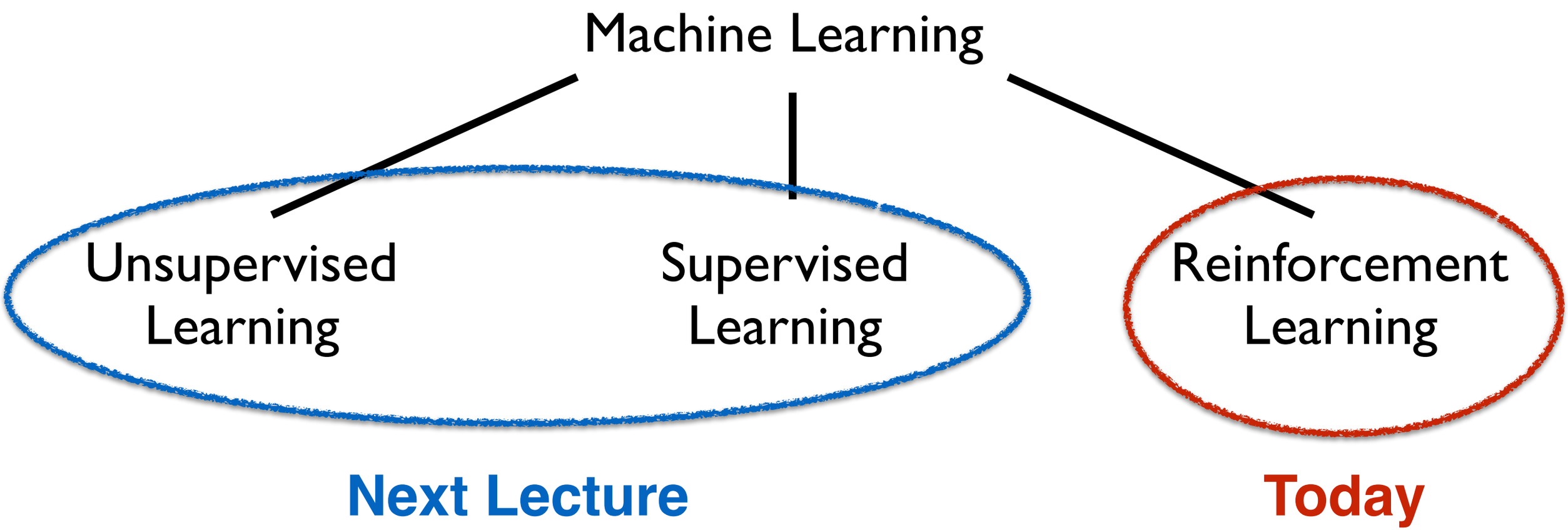
learn mapping from input-output pairs

Machine Learning



learn policy from **reward** for a given state action pair

Machine Learning

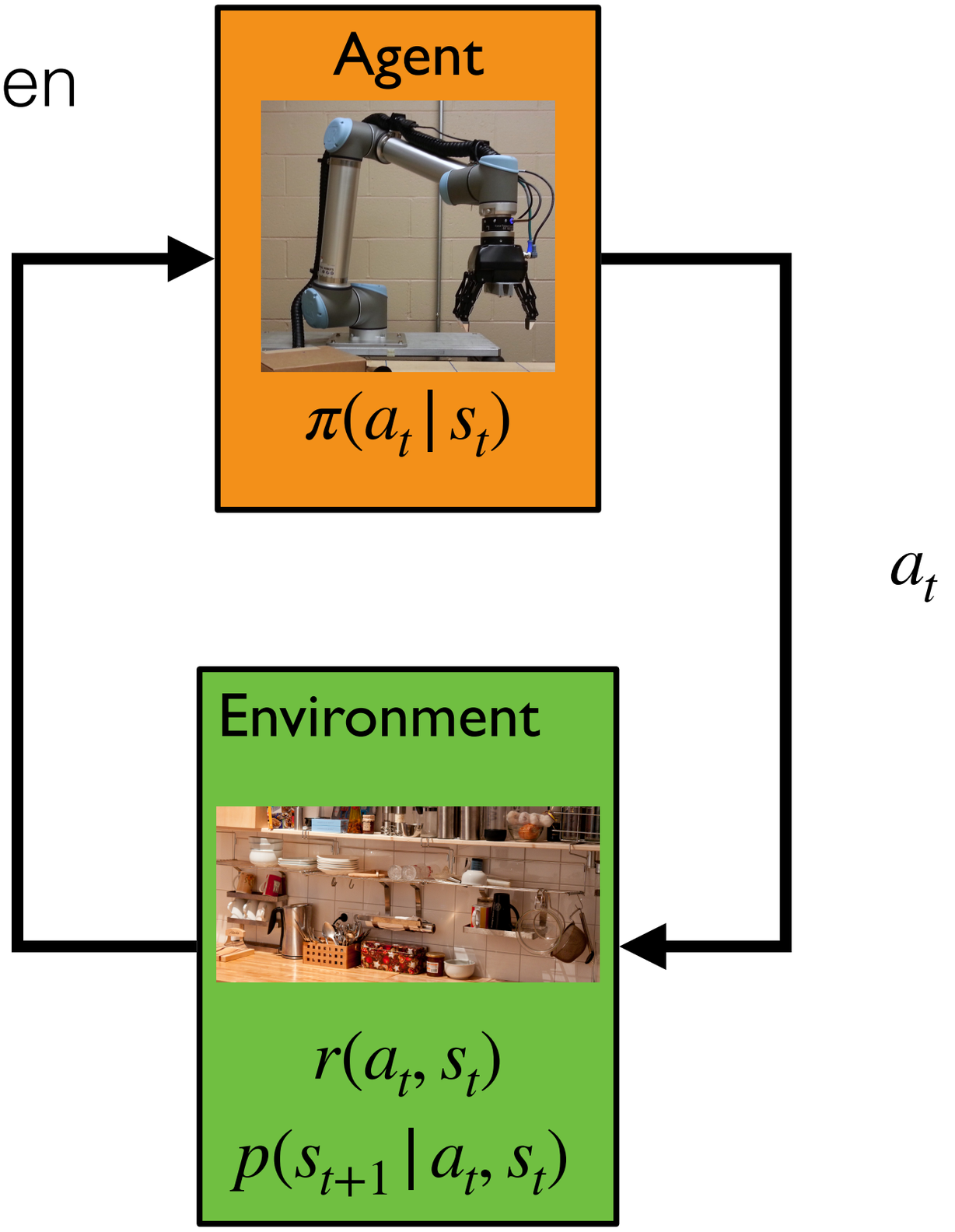


- Robots can acquire information from many sources
- Will be focusing on learning from own experiences

Markov Decision Processes

Markov Decision Processes

Task:
Cleaning up a kitchen



Markov Decision Processes

- State space

$$s_t \in \mathbb{S}$$

- Action space

$$a_t \in \mathbb{A}$$

- Transition probability

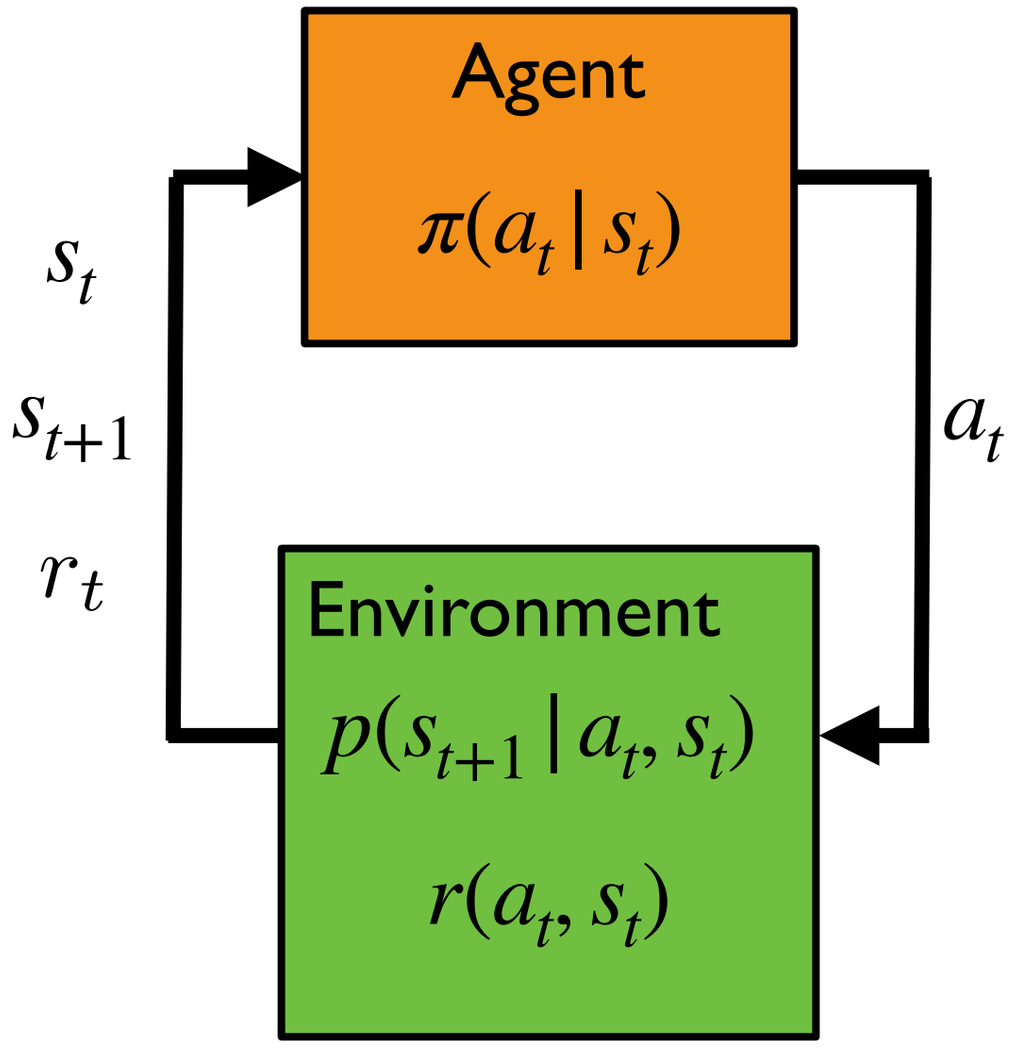
$$p(s_{t+1} \mid a_t, s_t)$$

- Reward function

$$r(a_t, s_t)$$

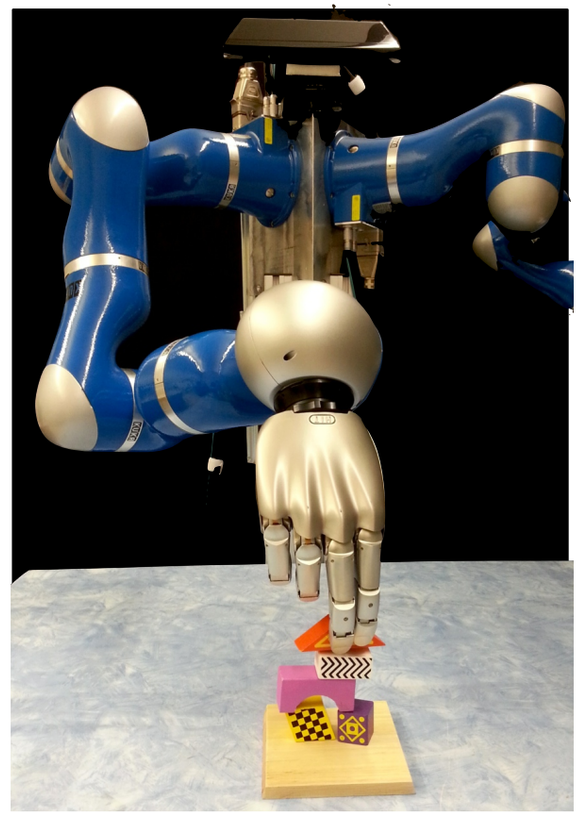
- Discount factor

$$0 \leq \gamma \leq 1$$



State and Markov Property

- State
  - positions of the hand and blocks
  - finger angles and contact forces
  - …



- Markov property
  - all future states are independent of the past states given the current state (the MDP is memoryless).

$$p(s_{t+1} \mid a_t, s_t, s_{t-1}, \ldots, s_1) = p(s_{t+1} \mid a_t, s_t)$$

  - transition distribution and policy depend only on current state

Markov Decision Process

- Goal:
  learn a policy π^* that maximises the expected return
- Policy

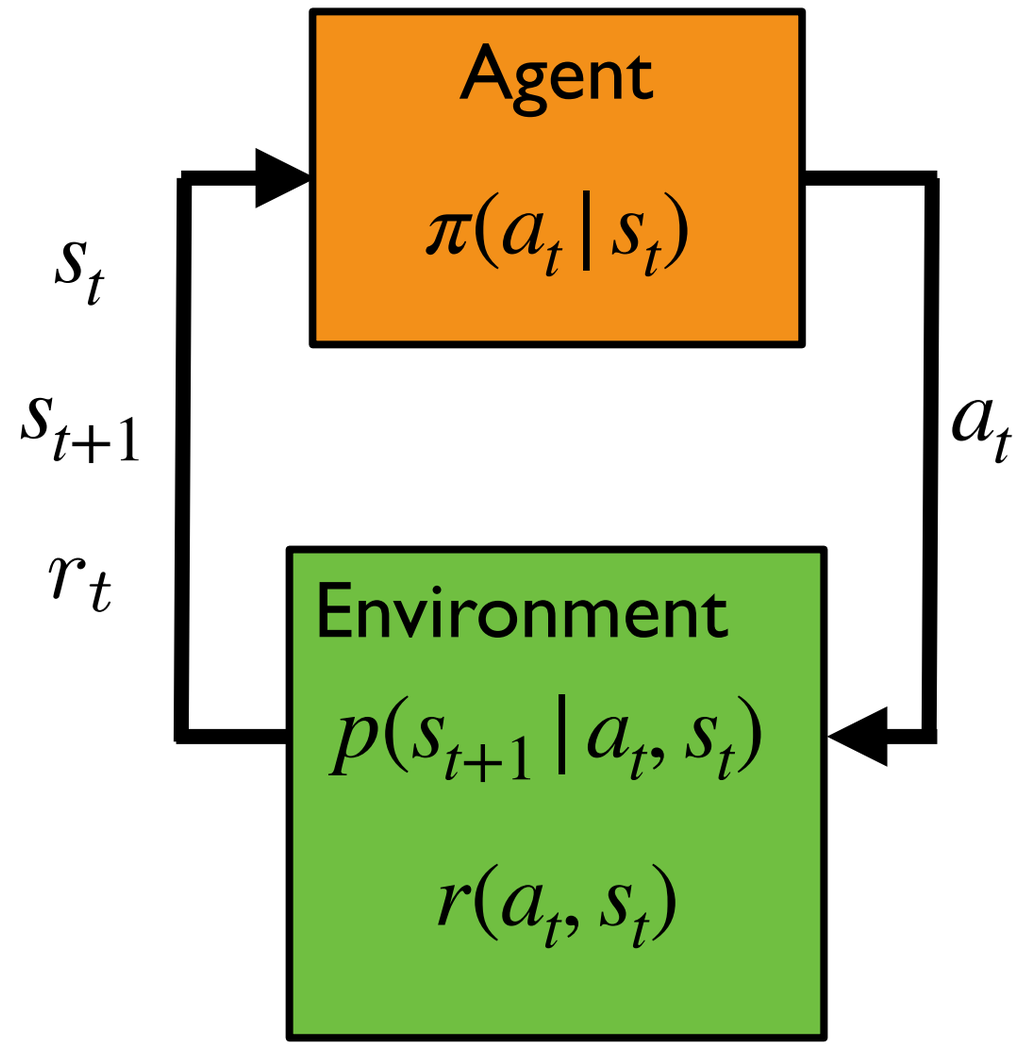
$$\pi(a_t \mid s_t)$$

- Expected Return

$$J(\pi) = E\left[\sum_{t=0}^{N} r(a_t, s_t)\right]$$

$$J(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t r(a_t, s_t)\right]$$

$$0 \leq \gamma < 1$$



Value Functions

Value Function

- Want the policy to generally choose actions that
  1. Give high immediate rewards
  2. Transition to good states for acquiring more rewards in future
- Need to determine how "good" a state is to be in

Value Function

- Want the policy to generally choose actions that
  1. Give high immediate rewards
  2. Transition to good states for acquiring more rewards in future
- Need to determine how "good" a state is to be in

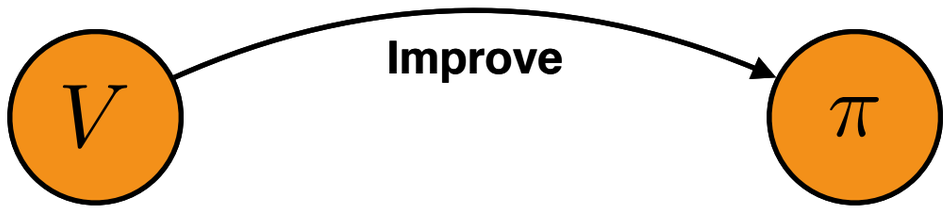
- Value function computes expected future discounted reward

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r(a_t, s_t) \,|\, s_0 = s\right]$$

  - given the initial state and current policy
- Value represents the reward-to-go given the policy

Generalized Policy Iteration

- Value function allows the robot to find better policies



- Policy improvement:
  - Given a value function, compute an improved policy

New Policy

$$\pi'(s_t) = \arg\max_a (r(a, s_t) + \gamma E[V^{\pi}(s_{t+1})])$$

Old Policy

Immediate Reward

"Good" Next State

Generalized Policy Iteration

- Lecture example:
  - V(sitting in front 3 rows)=100
  - V(sitting in back rows)= -50
- Who would stay in their seat? Who would move?

Generalized Policy Iteration

- Value function allows the robot to find better policies

- Policy evaluation:
  - Given a policy, compute the corresponding value function
  - Values will increase with more iterations for even better policies
- An optimal policy results in an optimal value function

$$\pi^* \rightarrow V^*(s) \qquad V^{\pi}(s) \leq V^*(s) \forall s$$

How to compute the value function?
Need estimates of V(s)

Monte Carlo Estimates

- The value function can be written as the expectation

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r(a_t, s_t) \,|\, s_0 = s\right]$$

- We can approximate this expectation based on samples

$$E[y] \approx n^{-1} \sum_{i=1}^{n} y_i$$

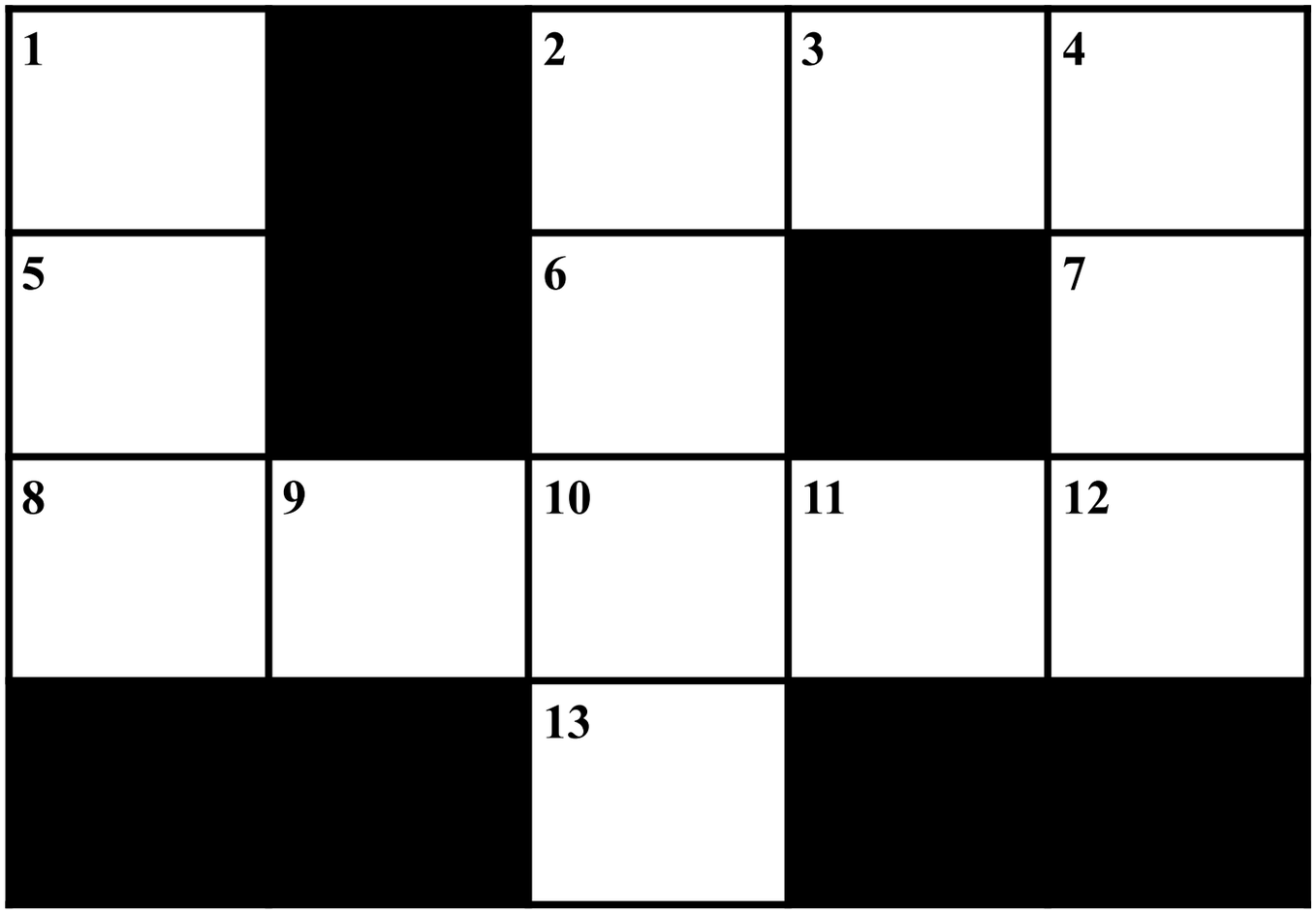
$$V^{\pi}(s) \approx n^{-1} \sum_{i=1}^{n} \sum_{t=0}^{\infty} \gamma^t r(a_{it}, s_{it})$$

  - where $s_{i0} = s$ and n is the number of times agent visited s

Gridworld Example

- Consider the following example problem

State space:

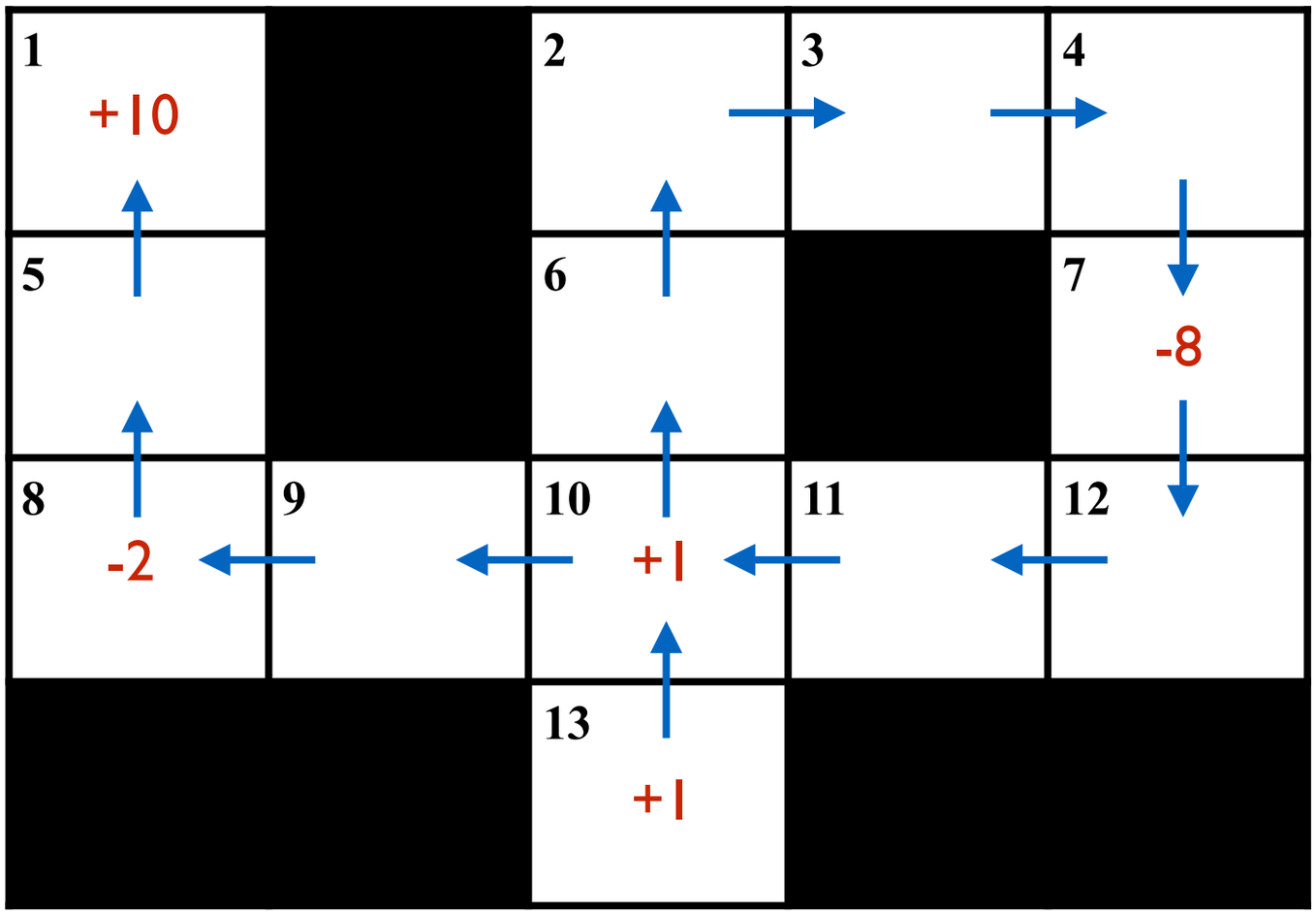


Action space:

left, right, up, down, and wait

Gridworld Example

- Consider the following rewards and current policy

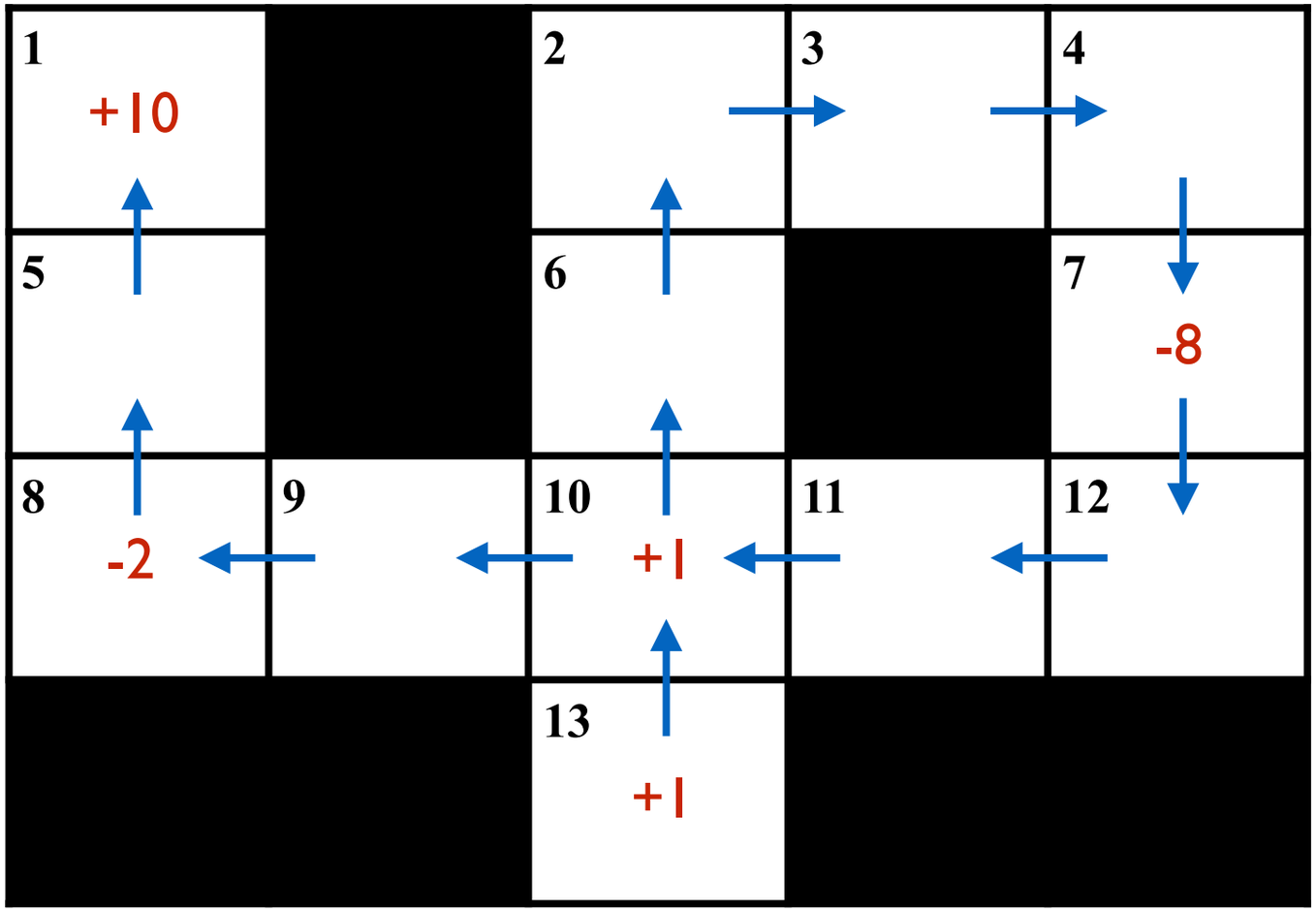


$$\gamma = 0.95$$

- What is the value of each state under this policy?

Monte Carlo Estimates

- Assume we have acquired the samples along the path



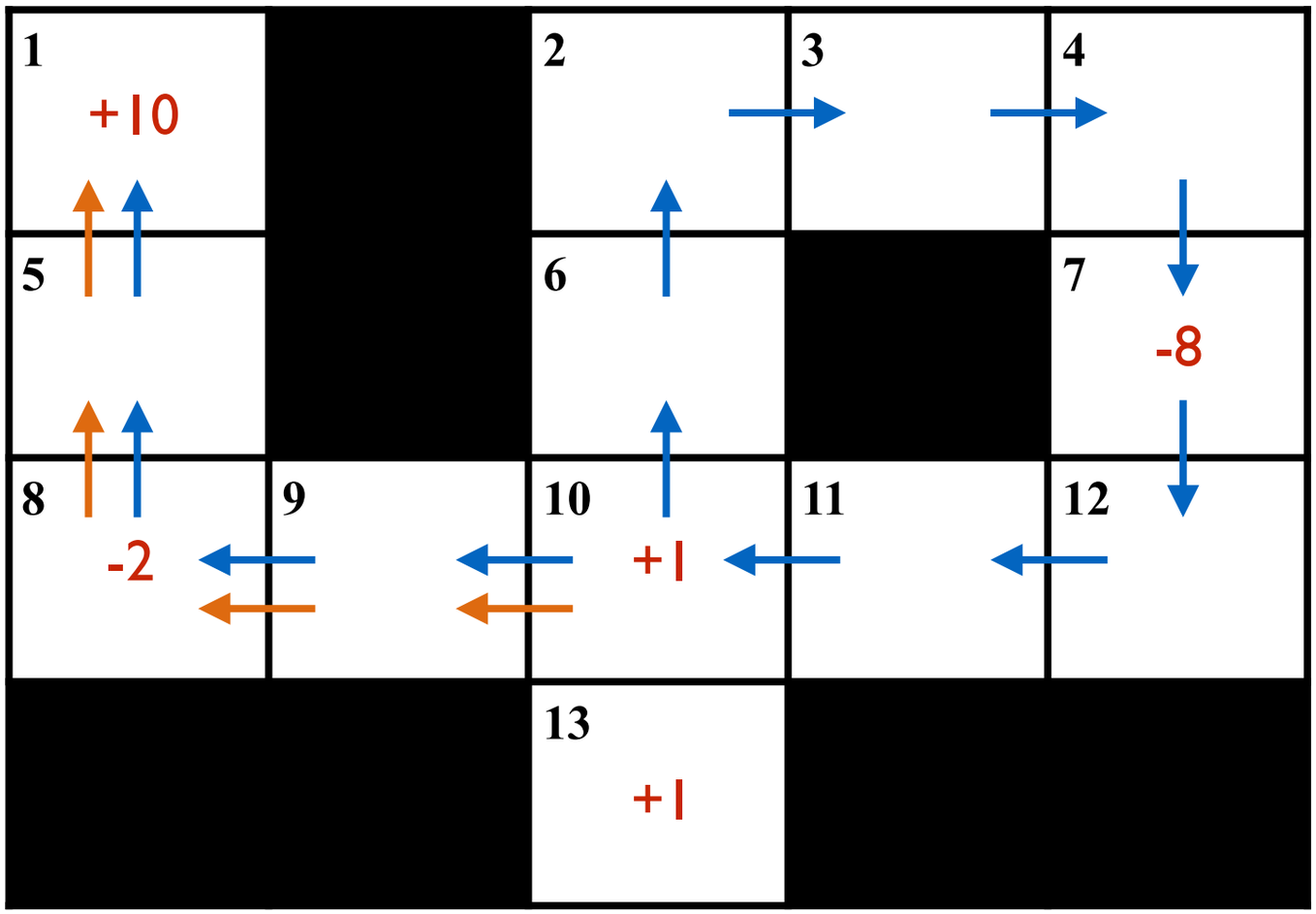
$$\gamma = 0.95$$

$$V^{\pi}(s = 13) = 1 + 0.95^{1}1 + 0.95^{6}(-8) + 0.95^{9}1 + 0.95^{11}(-2) + 0.95^{13}10$$

$$V^{\pi}(s = 13) = 0.695$$

Monte Carlo Estimates

- Assume we have acquired the samples along the path



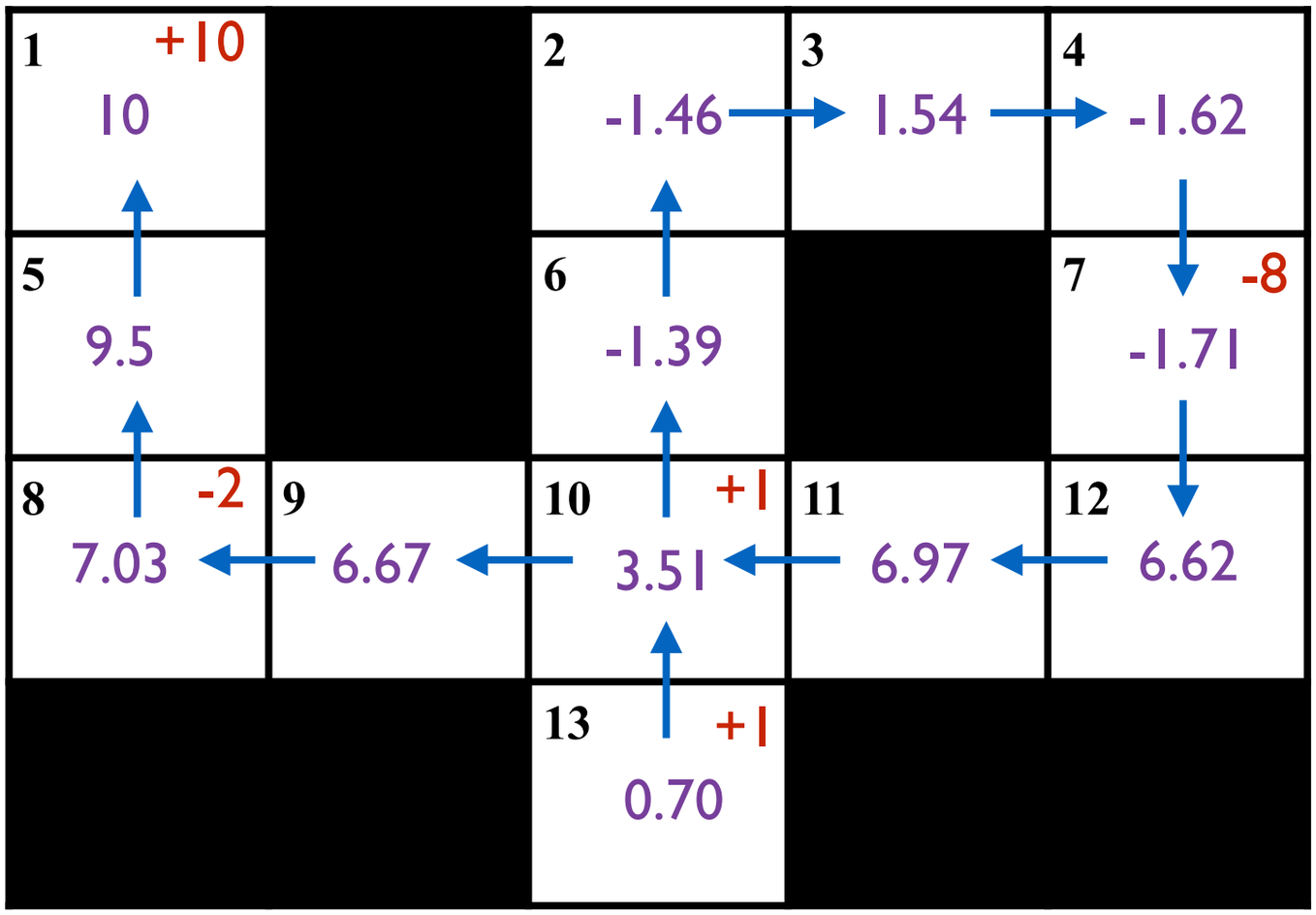
$$\gamma = 0.95$$

$$V^{\pi}(s=10) = \frac{1}{2}(1 + 0.95^5(-8) + 0.95^8 1 + 0.95^{10}(-2) + 0.95^{12}10)$$

$$+\frac{1}{2}(1 + 0.95^2(-2) + 0.95^4 10) = -0.16 + 3.67 = 3.51$$

Monte Carlo Estimates

- Resulting values:



$$\gamma = 0.95$$

- Could use the new estimates to update policy for state 10
- Monte Carlo has large variances for less constrained problems
- Low value for state 13 given transition to state 10's value

Temporal Difference Learning

- The value function can be written in recursive form

$$V^{\pi}(s) = E[r(a_0, s_0) + \gamma V^{\pi}(s') \,|\, s_0 = s]$$

next state

- Estimate of value $V^{\pi}(s_t)$ is thus given by single step

$$r_t + \gamma V^{\pi}(s_{t+1})$$

- Bootstrap estimate using the current value estimates i.e., value of next state is just an estimate

Temporal Difference Learning

- Approximate value using a weighted average

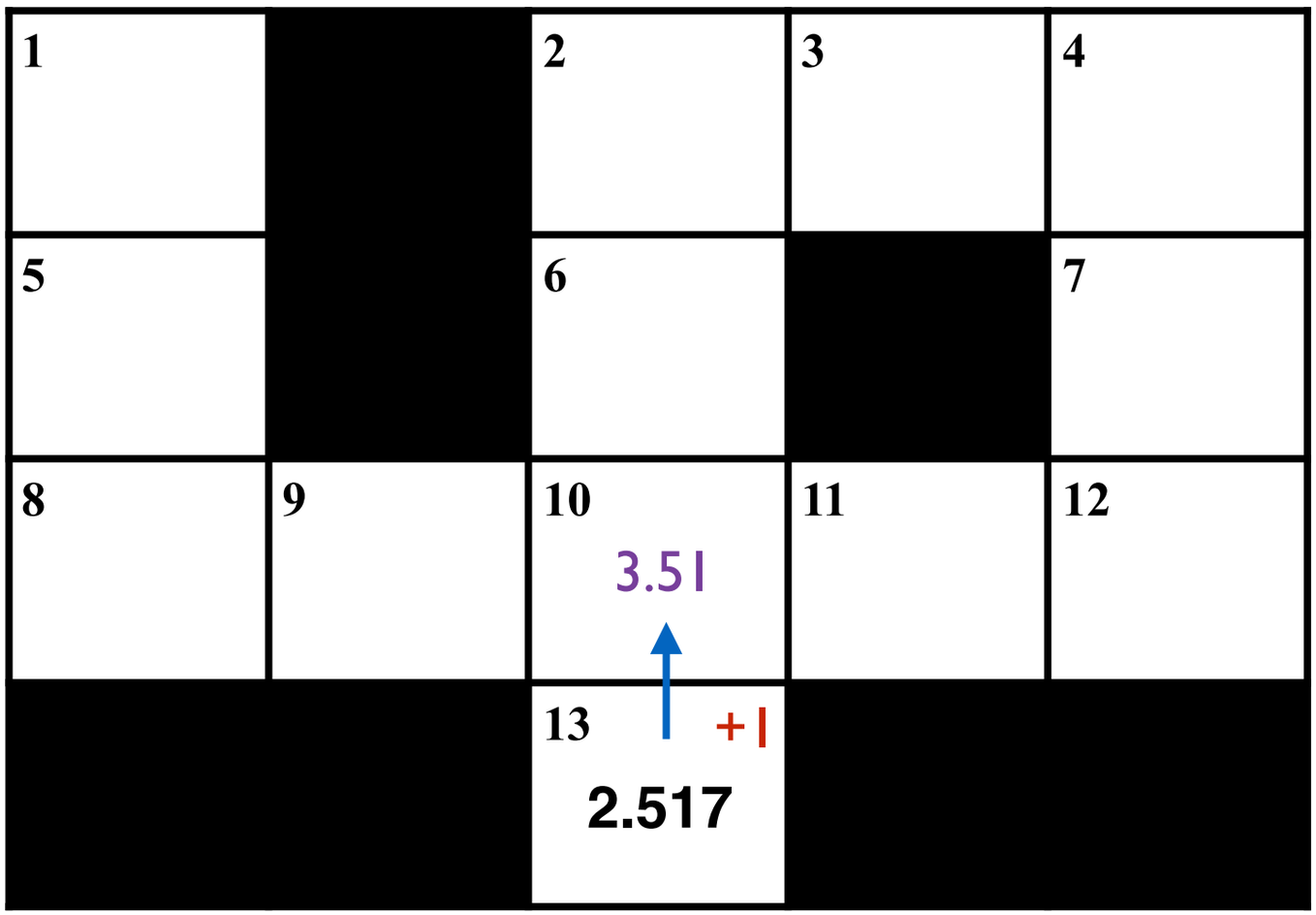
$$V^{\pi}_{\text{new}}(s_t) = (1-\alpha)V^{\pi}(s_t) + \alpha(r_t + \gamma V^{\pi}(s_{t+1}))$$

$$V^{\pi}_{\text{new}}(s_t) = V^{\pi}(s_t) + \underbrace{\alpha(r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))}_{\text{TD Error}}$$

- TD error between expected and estimated values
- Expected TD error will tend towards zero

Temporal Difference Example

- Lets consider the update for this one step



$$\gamma = 0.95$$

$$\alpha = 0.5$$

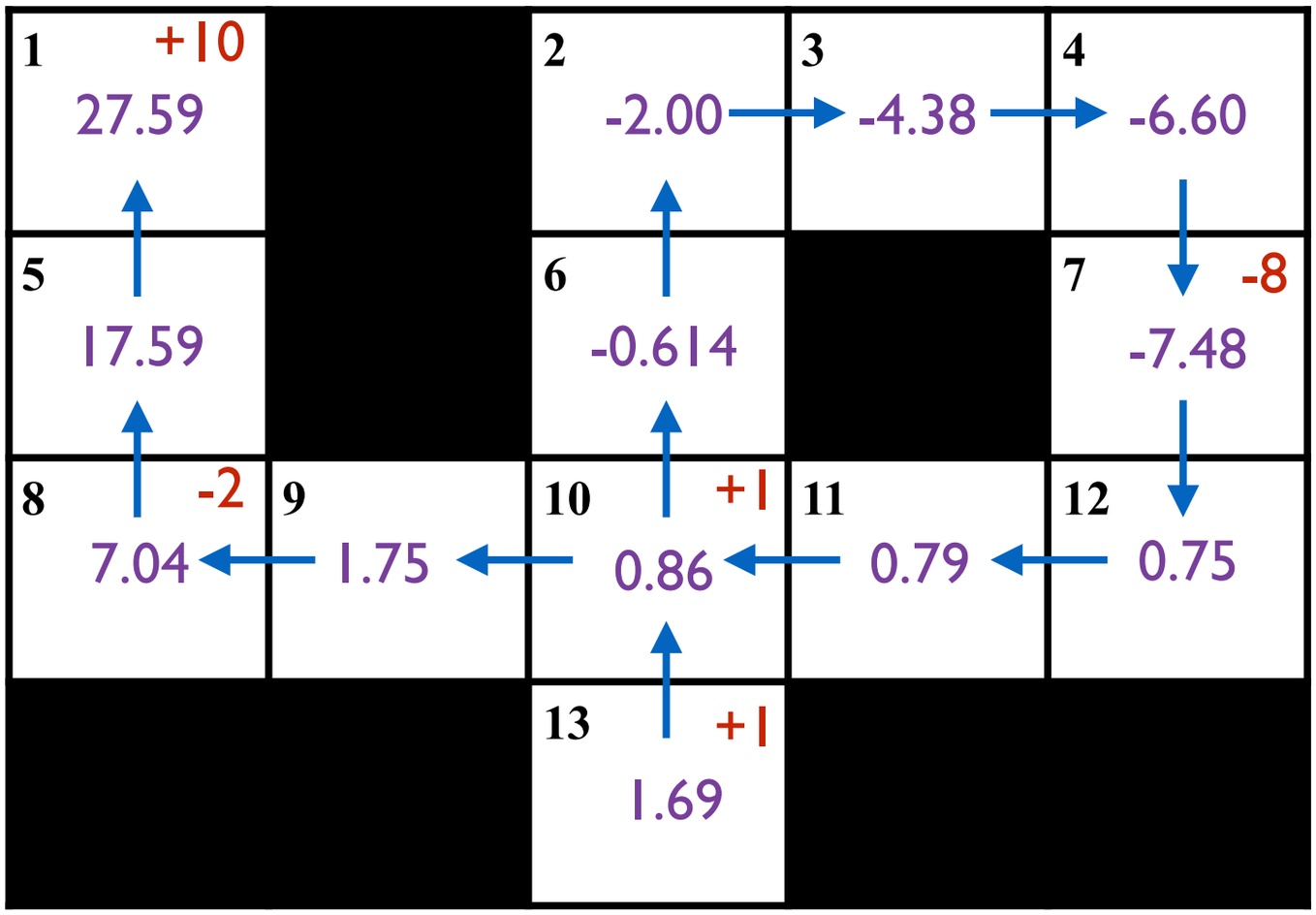
$$V^{\pi}_{\text{new}}(s_t) = V^{\pi}(s_t) + \alpha(r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$$

$$V^{\pi}_{\text{new}}(s = 13) = 0.7 + 0.5(1 + 0.95(3.51) - 0.7)$$

$$V^{\pi}_{\text{new}}(s = 13) = 0.7 + 0.5(3.6345) = 2.517$$

Temporal Difference Example

- After looping through the samples ten times



$$\gamma = 0.95$$

- Can update policy along the way
- Estimates are more consistent with lower variance
- Takes a while to propagate distant rewards

State-Action Values

- Policy improvements currently rely on transition model

$p(x_{t+1}|u_t, x_t)$

$$\pi'(x_t) = \arg\max_u \left(r(u, x_t) + \gamma E\left[V^\pi(x_{t+1})\right]\right)$$

- Learn state-action values directly

$$Q^\pi(x, u) = E\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, u_t) | x_0 = x, u_0 = u\right]$$

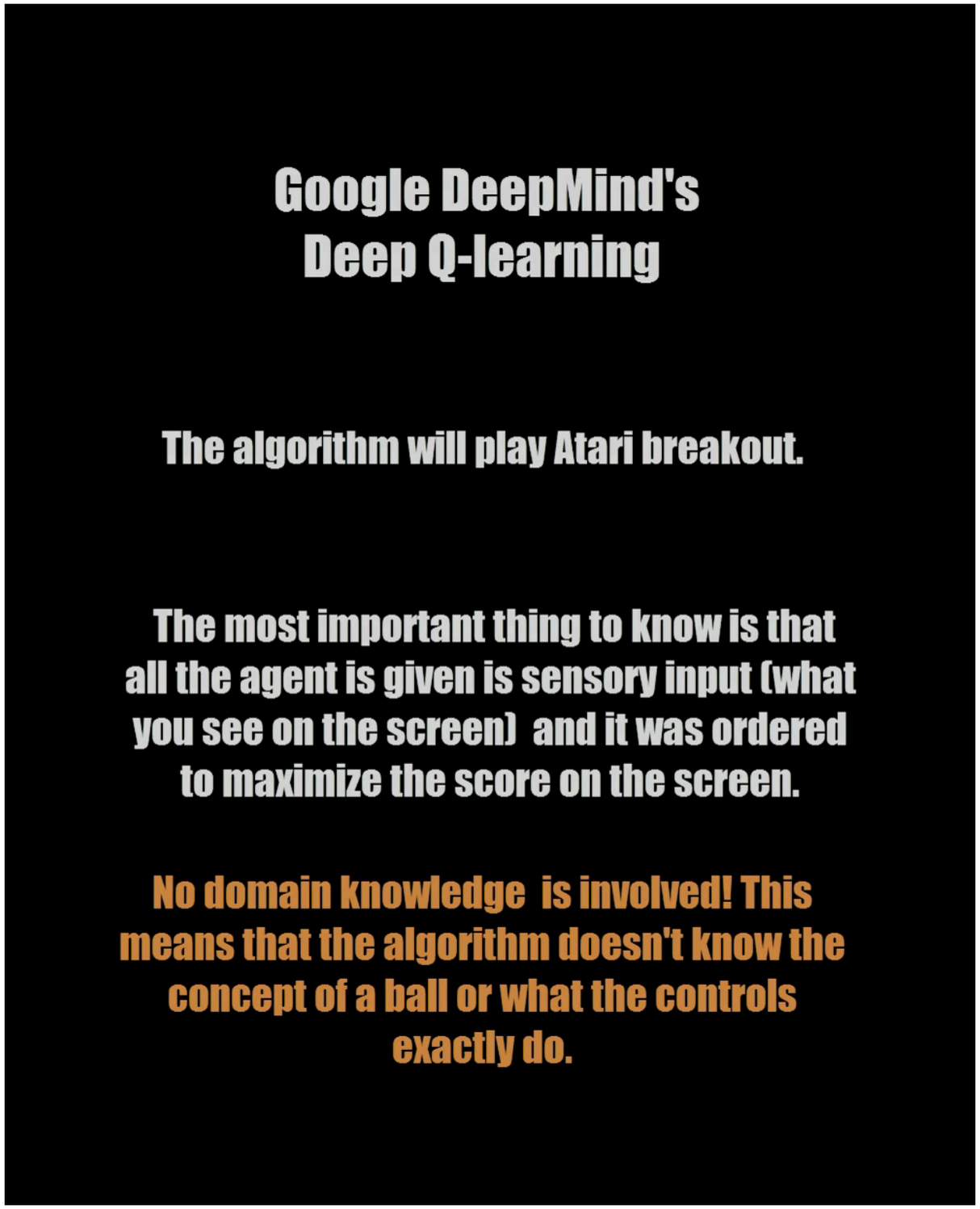
$$Q^\pi(x, u) = r(x, u) + \gamma E\left[V(x')\right]$$

  - Reward and transition to next state are implicit

- Policy improvement is then given by

$$\pi'(x) = \arg\max_u Q^\pi(x, u)$$

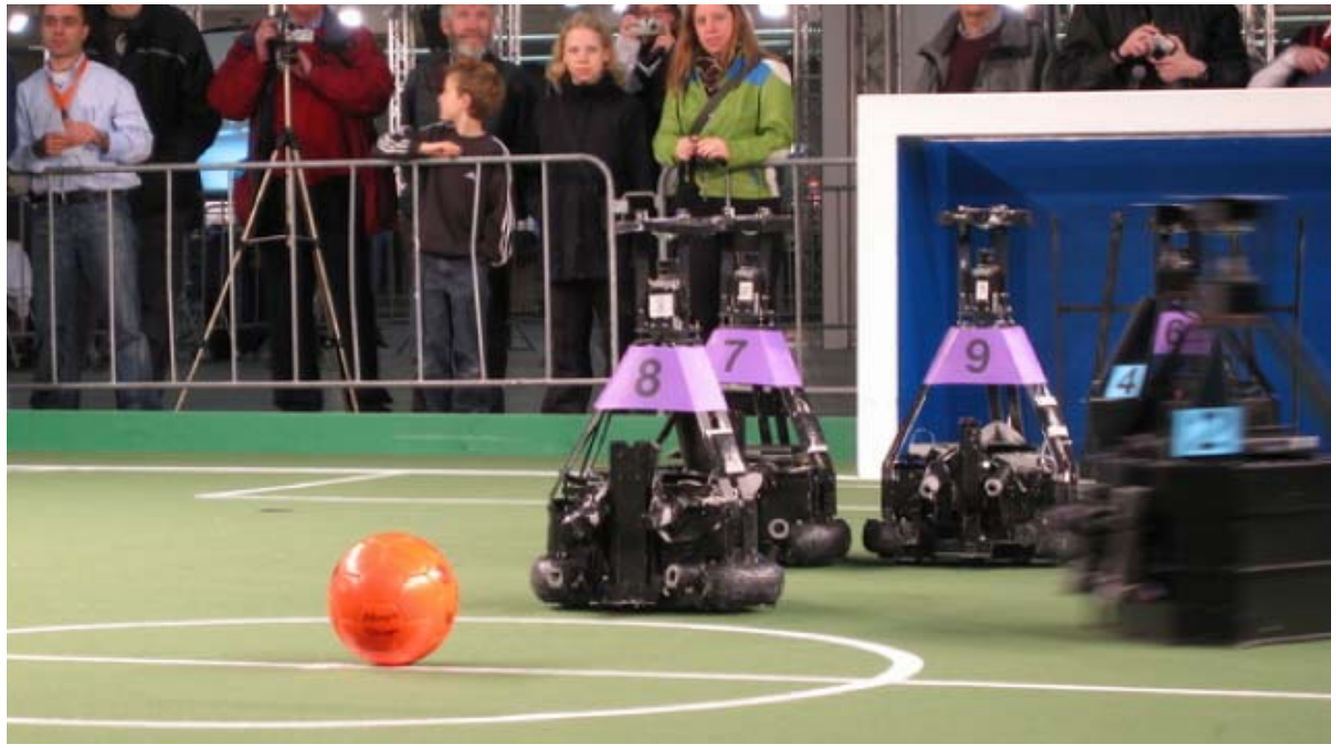
Atari Example

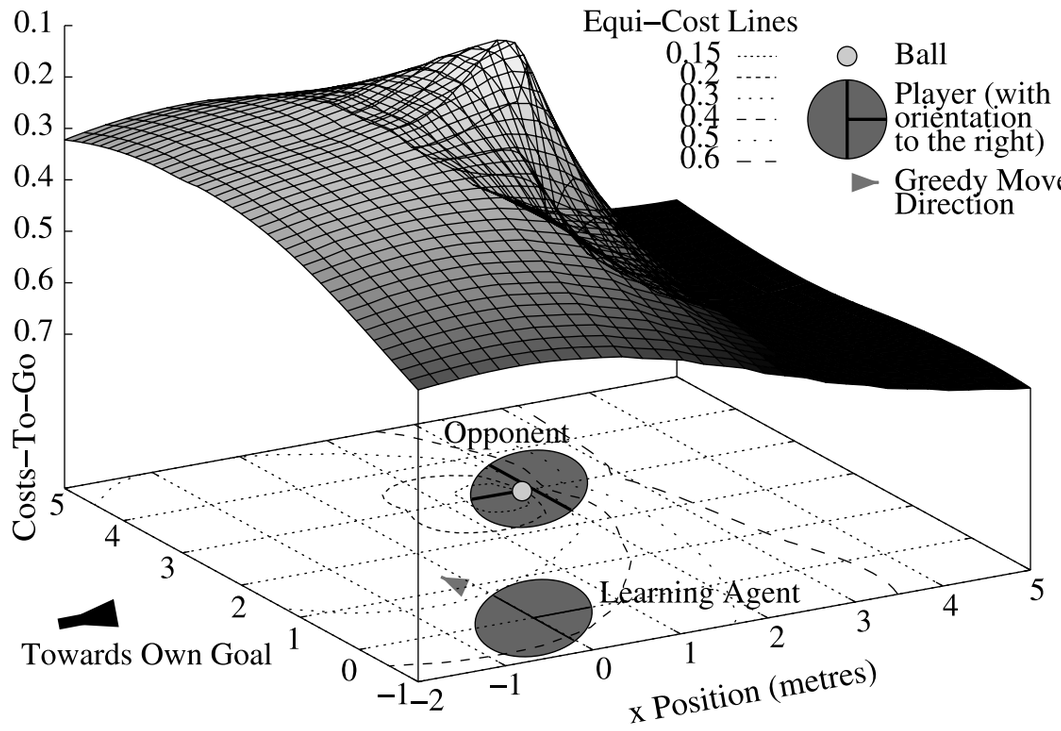


V. Minh et al., Playing Atari with Deep Reinforcement Learning, 2013
Two Minute Papers,Google DeepMind's Deep Q-learning playing Atari Breakout, Youtube,, 2015

Robot Example

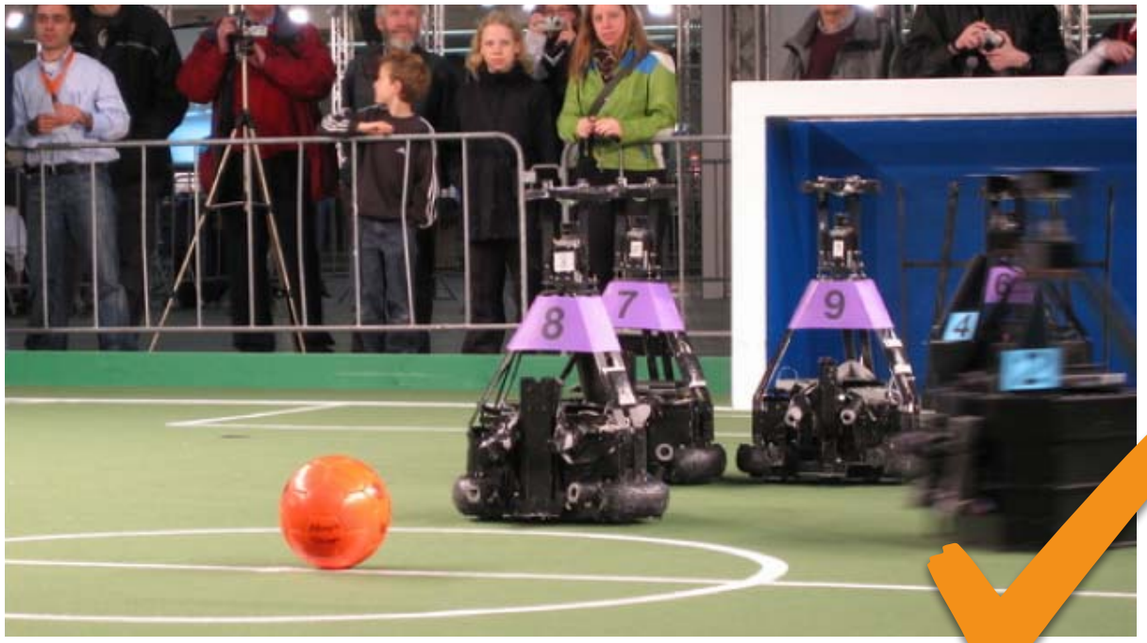
- Q-Learning for learning soccer skills

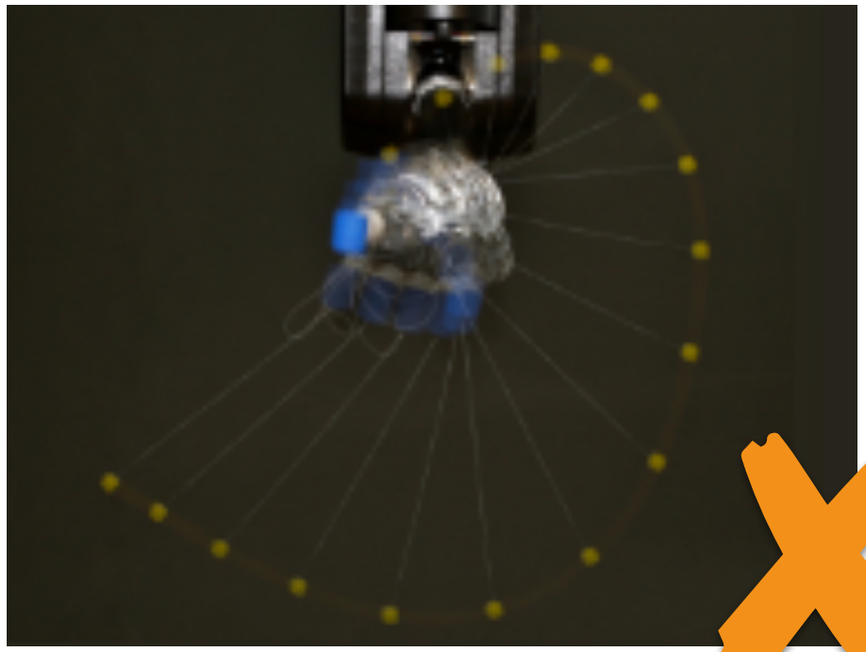




- Robot learns skills such as dribbling and duelling
- Figure shows slice of 9D-input value function

Reinforcement Learning for Robotics





- Value function methods are powerful, but can be difficult to apply to high-dimensional state spaces
- Complex skills often learned using policy search methods

Policy Search Methods

Parameterized Policies

- Policies often have a parameterized form
  - Gaussian linear feedback policy

$$a_t \sim \mathcal{N}(-\Theta s_t, \sigma^2)$$

  - Desired trajectory basis functions

$$s_t^d = \sum_i \theta_i \mathcal{N}(\mu_i - t, h^2)$$

  - Neural network controller

$$a_t \sim \mathcal{N}(f_{NN}(s_t; \theta), \sigma^2)$$

- May be deterministic or stochastic
- Define a family of potential controller solutions

Problem Statement

- Executing a policy results in a sample trajectory

$$\tau = \{s_0, a_0, \ldots s_T, a_T, s_{T+1}\}$$

- Each trajectory provides the robot with a return

$$R(\tau) = \sum_{t=0}^{T} r(s_t, a_t)$$

- Goal: find policy that maximizes the expected return

$$J(\theta) = \int_{\mathbb{T}} p(\tau \,|\, \pi_\theta) R(\tau) \mathrm{d}\tau$$

Find policy parameters that maximize the expected return

Problem Statement

- The goal is to compute $\arg\max_\theta J(\theta)$
- Finding the global optimum is intractable
  - Complicated dynamics and policies
  - Non-convex reward landscape
  - High-dimensional state space
  - No closed-form solutions
- Search (locally) for good policy parameters
- Can use gradient ascent to find (good) local optimum

$$\theta_{i+1} = \theta_i + \beta \nabla_\theta J(\theta)$$

- How to compute estimate of the gradient?

Finite Difference Method

- Finite difference method procedure:

1. Initialize policy with some initial parameters $\theta_{i=0}$
2. Sample neighborhood of parameters $\tilde{\theta}_{ij} \quad j \in \{1, ..., n\}$
3. Evaluate parameter settings to estimate returns $J(\theta_i) \quad J(\tilde{\theta}_{ij})$
4. Compute finite difference gradient estimate g_{fd}
5. Update parameters using gradient ascent $\theta_{i+1} = \theta_i + \beta g_{\text{fd}}$
6. Repeat from step 2

- Final performance depends on good initial parameters

Finite Difference Method

- Sample n local parameters from grid or stochastically

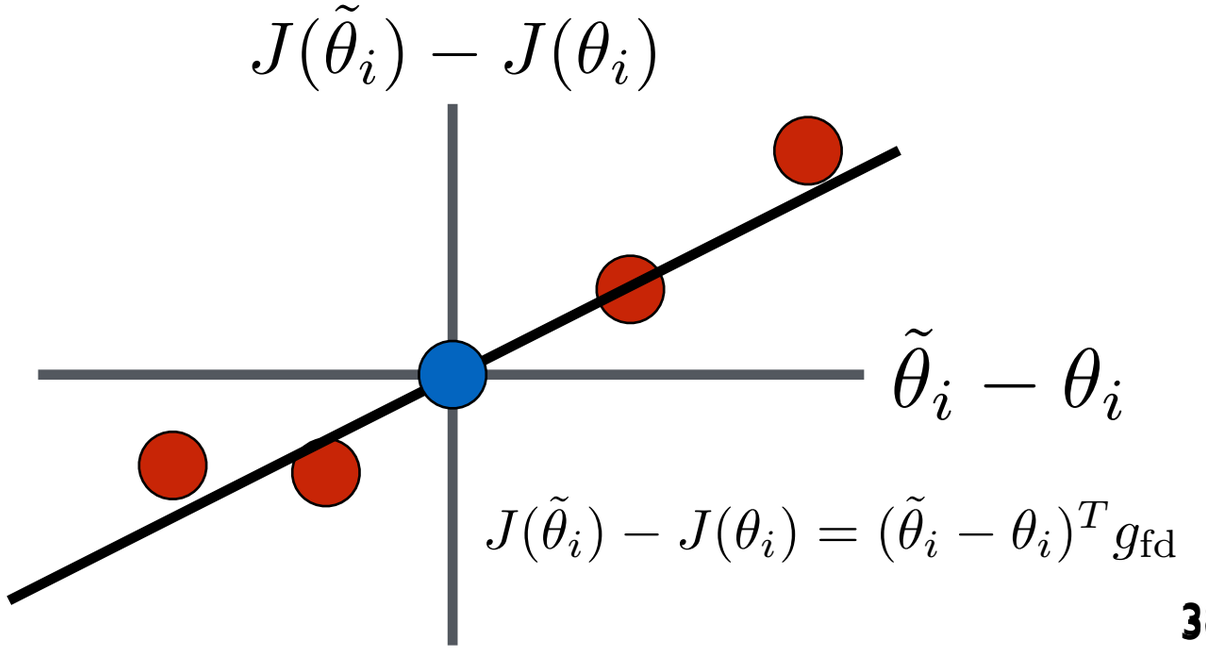
$$\tilde{\theta}_{ij} = \theta_i + \delta \qquad \tilde{\theta}_{ij} \sim \mathcal{N}(\theta_i, \Sigma)$$

- Estimate gradient using linear regression on differences

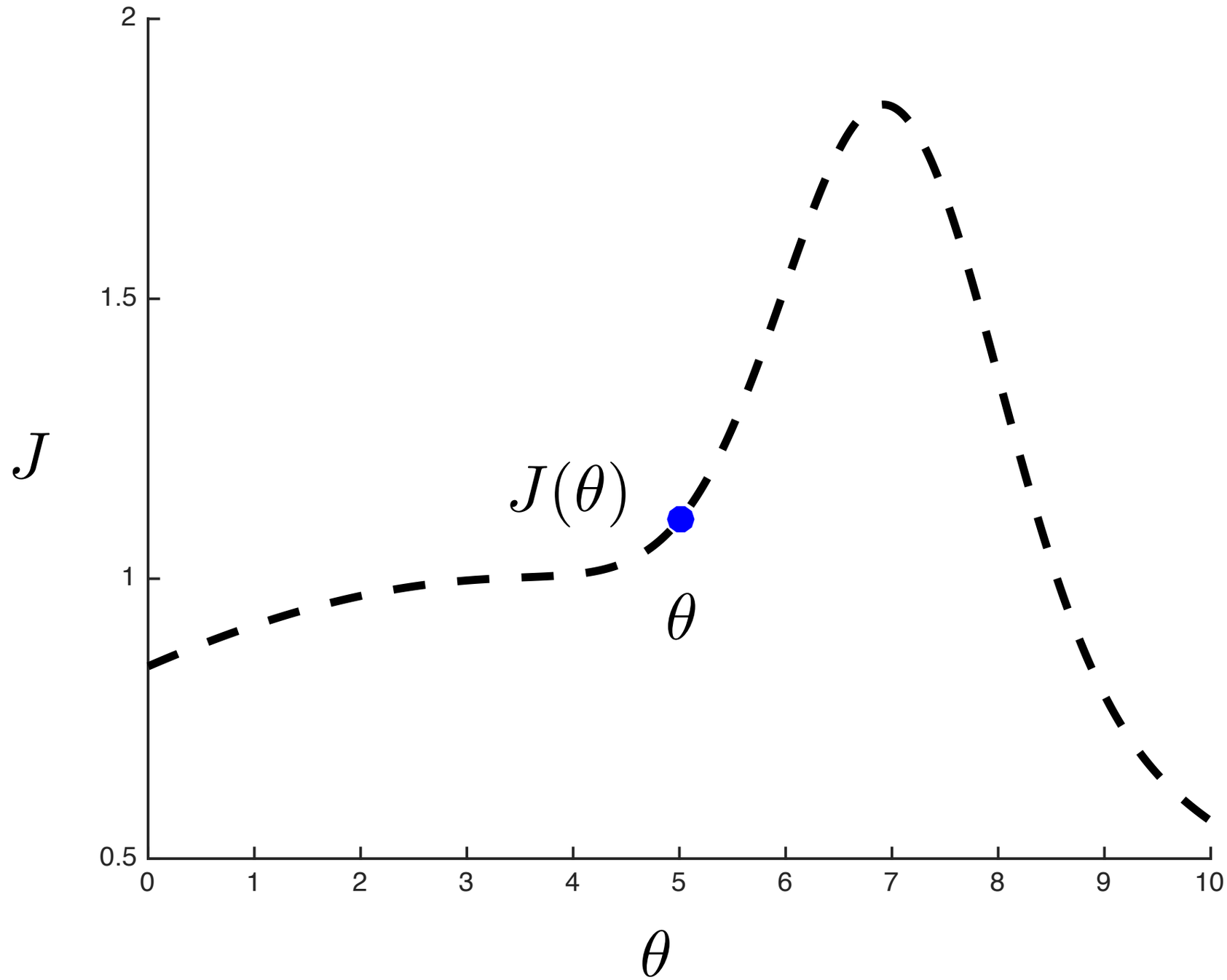
$$\widehat{\Theta} = \begin{bmatrix} | & & | \\ \tilde{\theta}_{i1} - \theta_i & \dots & \tilde{\theta}_{in} - \theta_i \\ | & & | \end{bmatrix} \qquad \widehat{J} = \begin{bmatrix} J(\tilde{\theta}_{i1}) - J(\theta_i) \\ \vdots \\ J(\tilde{\theta}_{in}) - J(\theta_i) \end{bmatrix}$$

$$g_{\text{fd}} = (\widehat{\Theta}\widehat{\Theta}^T + \lambda I)^{-1}\widehat{\Theta}\widehat{J}$$

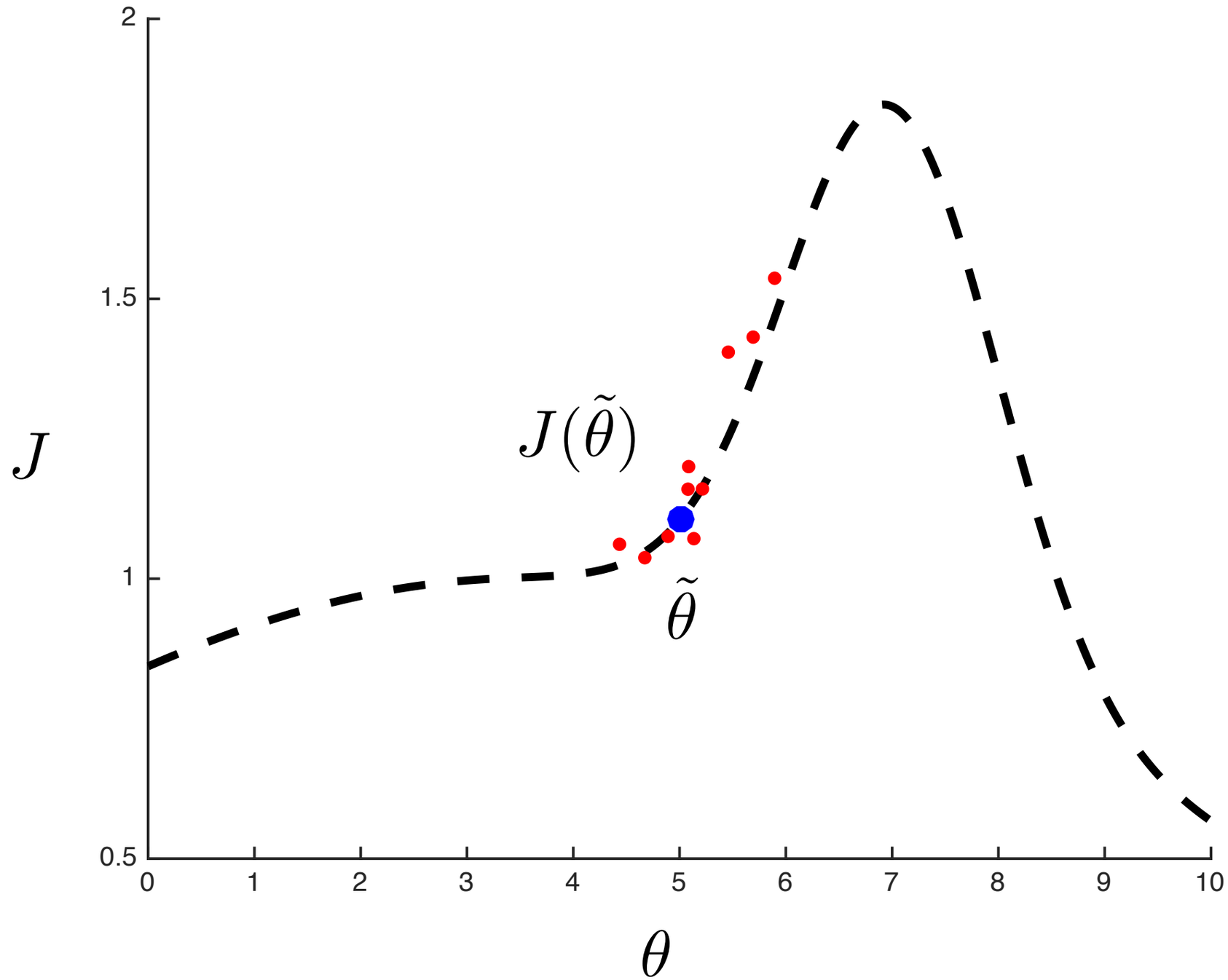
$$\theta_{i+1} = \theta_i + \beta g_{\text{fd}}$$



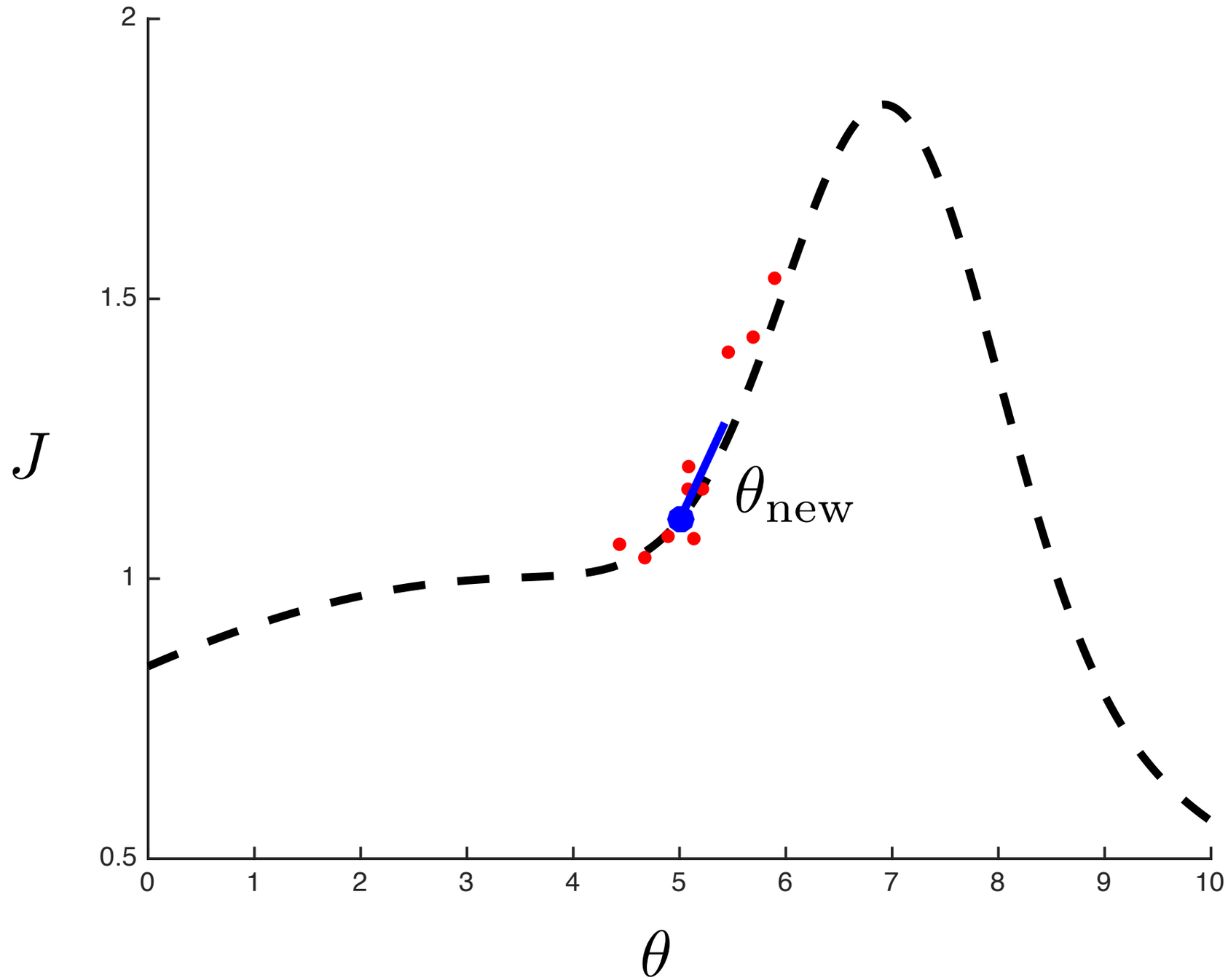
Finite Difference Example



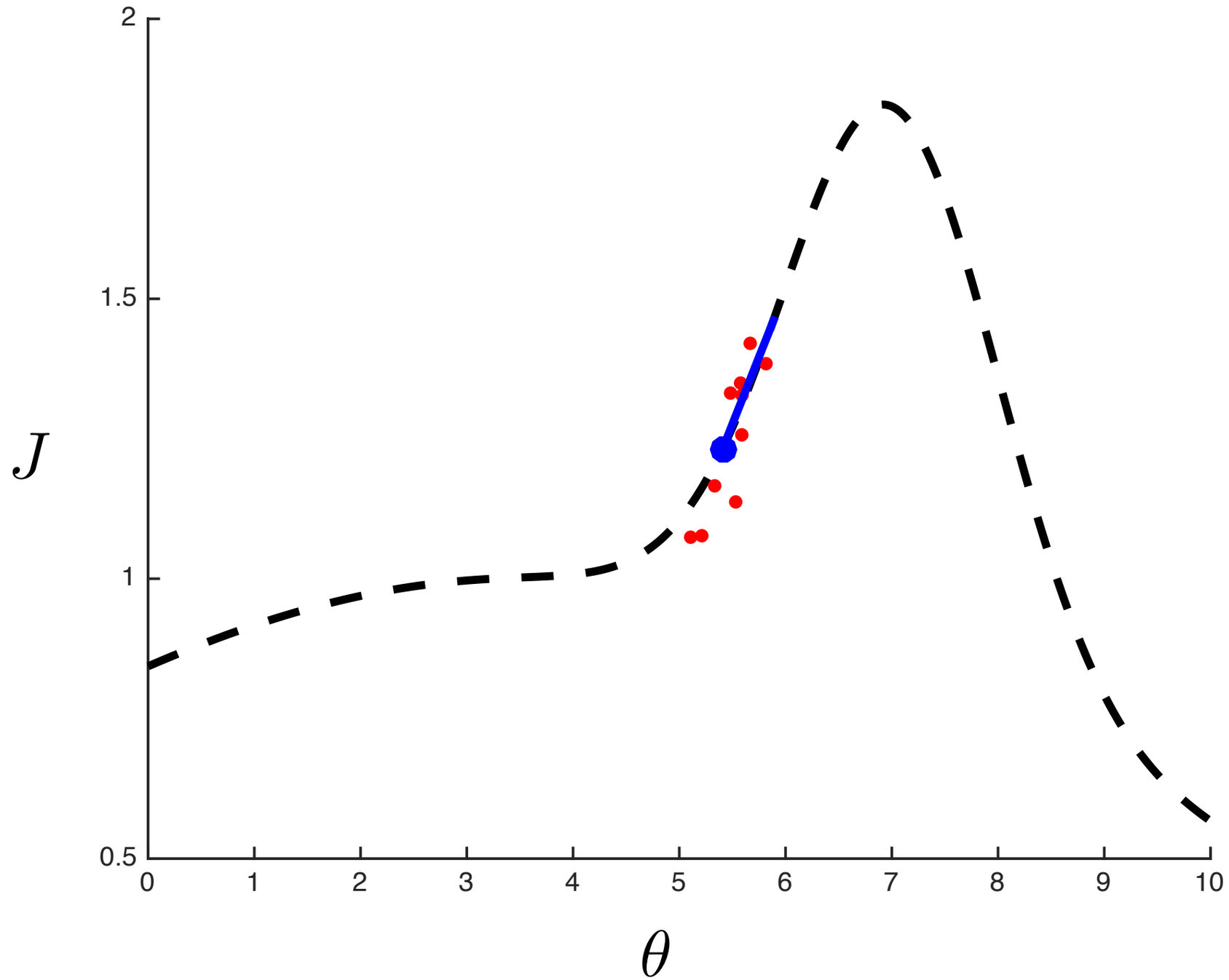
Finite Difference Example



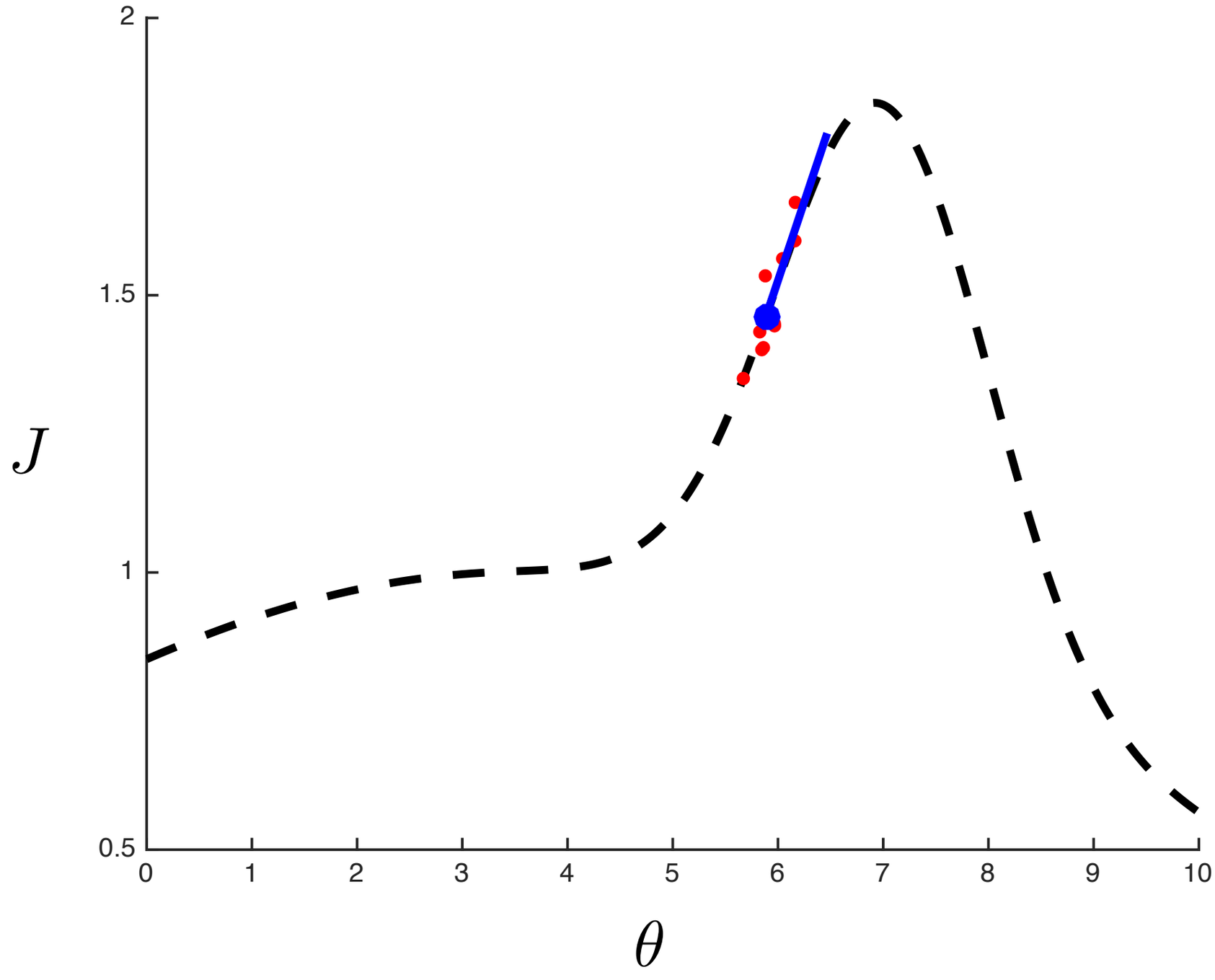
Finite Difference Example



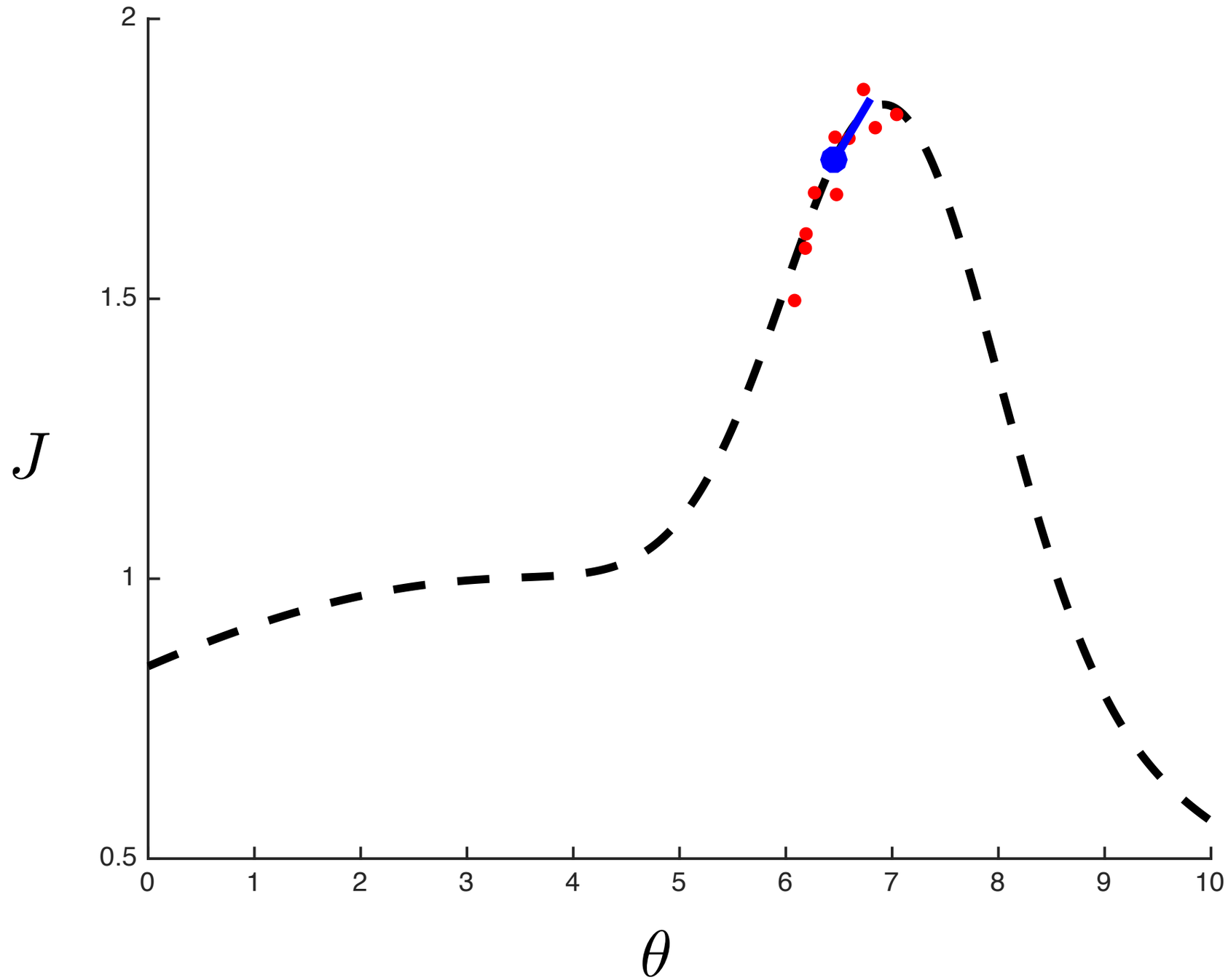
Finite Difference Example



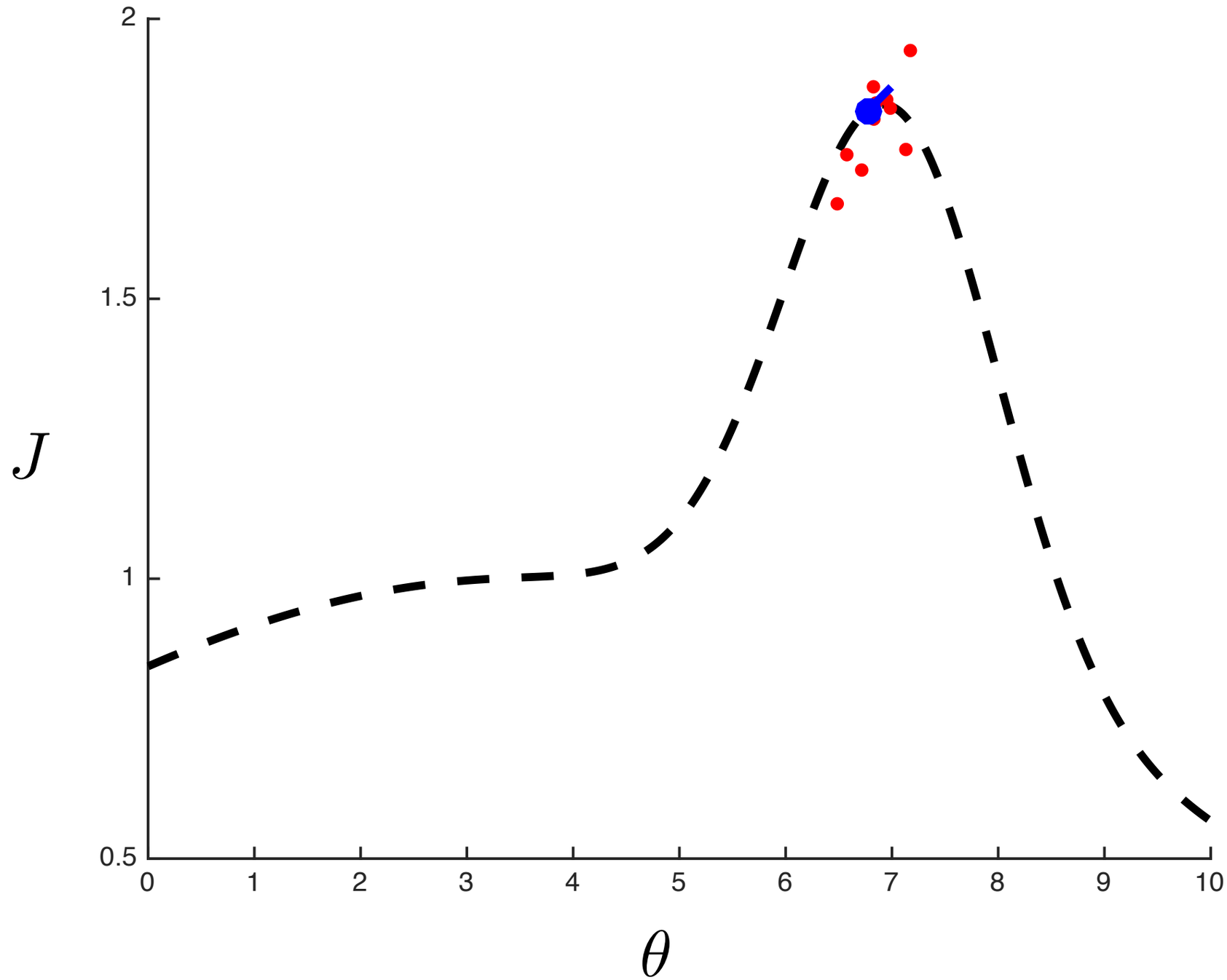
Finite Difference Example



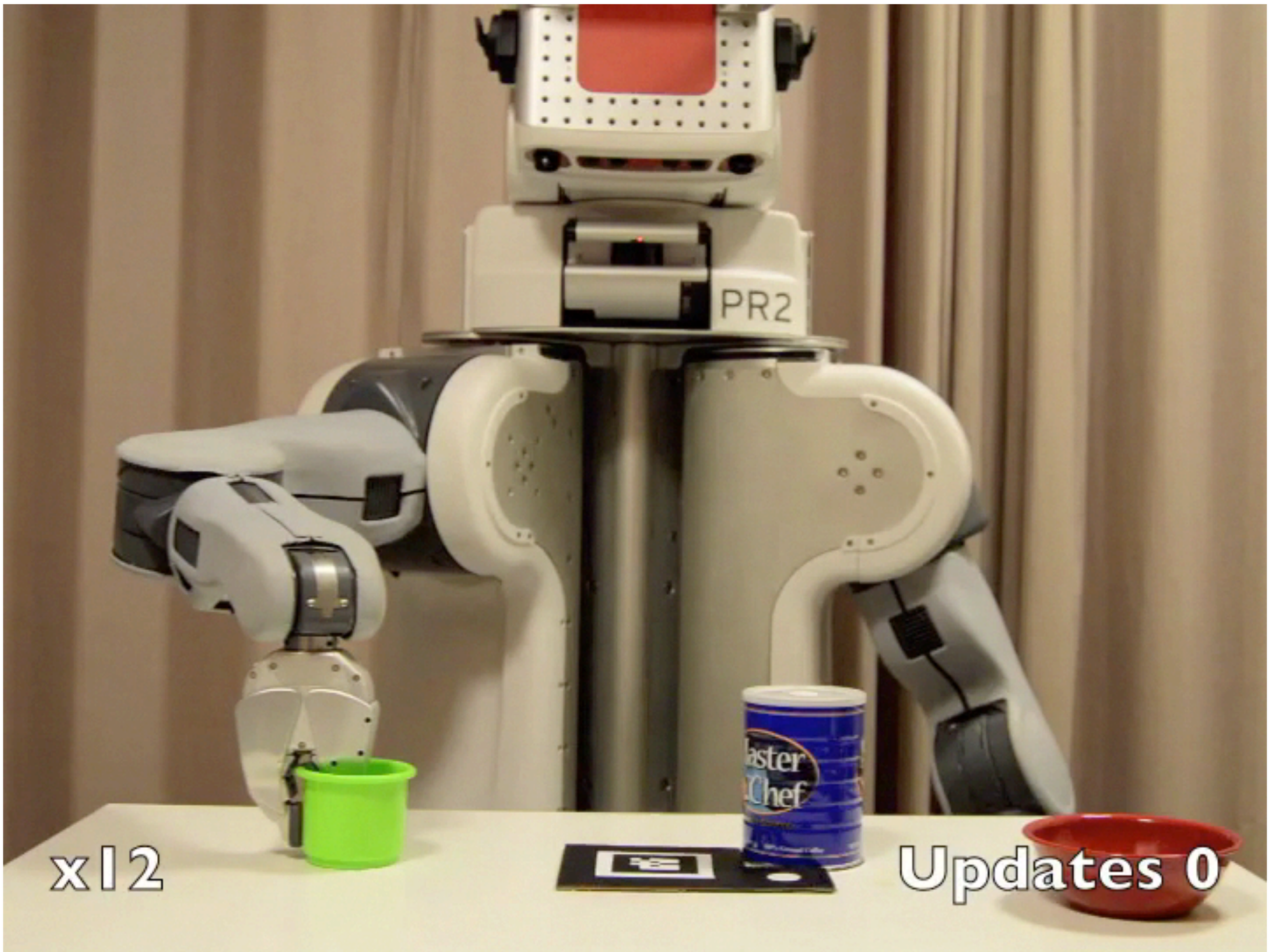
Finite Difference Example



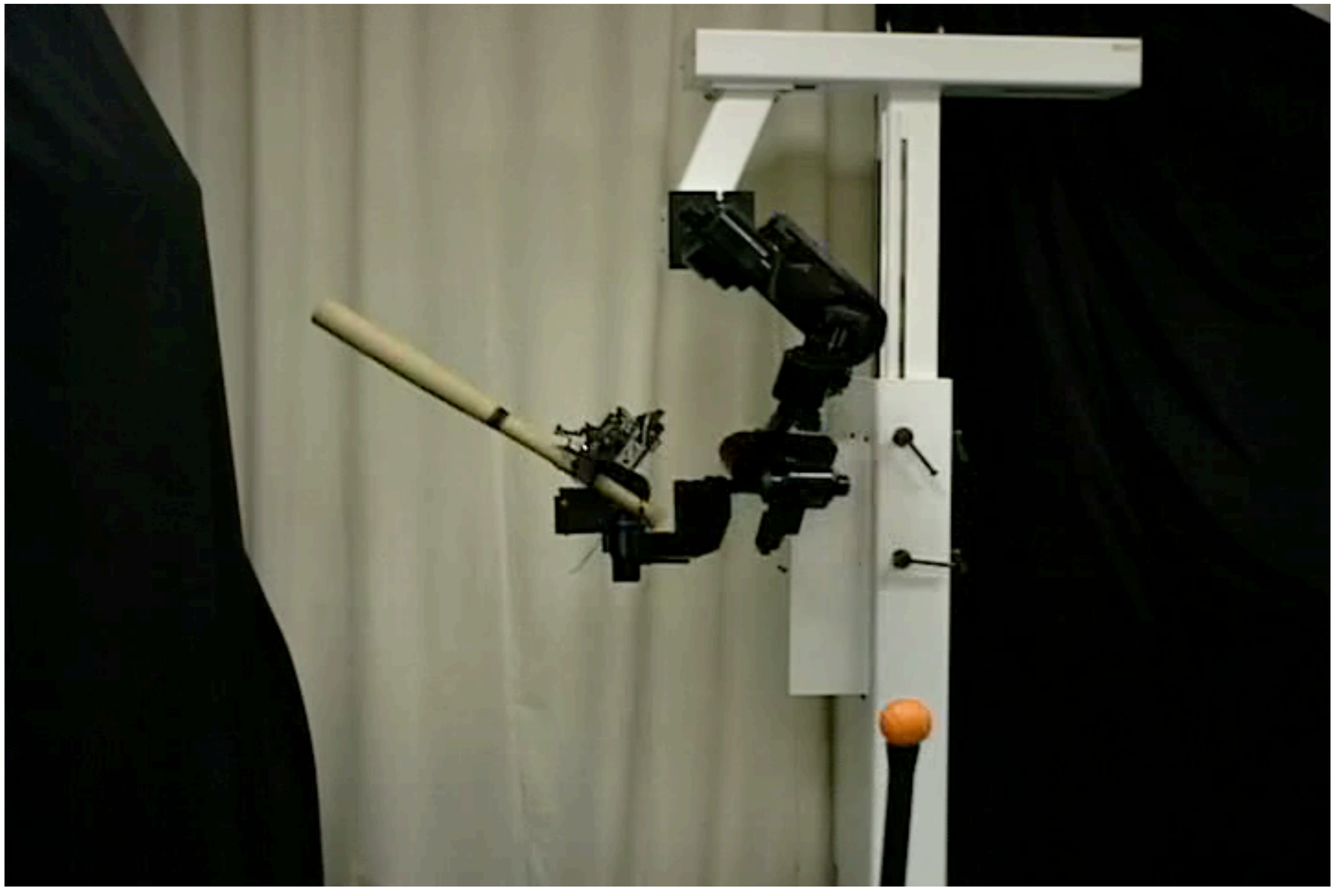
Finite Difference Example



REPS Example



Natural Actor Critic Example



Options and Hierarchical Policies

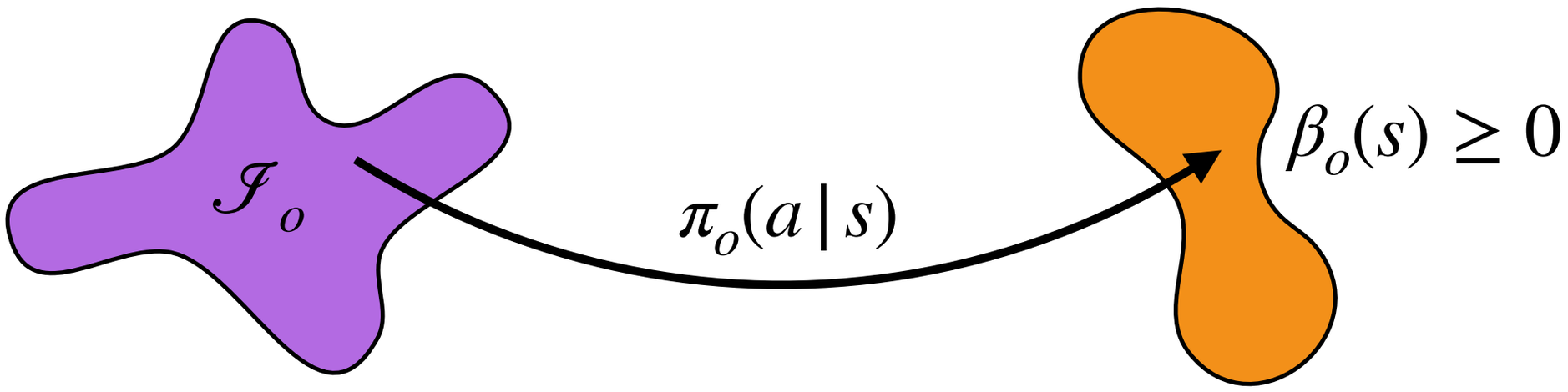
Options

- Often need to perform certain subtasks many times
  - Grasping objects, moving between rooms…
- Always reasoning about lowest-level actions is inefficient
  - Joint torque controller for cleaning an entire building…

- Learn Options as sub-policies for subtasks
  - Option policy $\pi_o(a \mid s)$
  - Initiation set $\mathcal{I}_o \subseteq \mathbb{S}$
  - Termination conditions $\beta_o(s)$

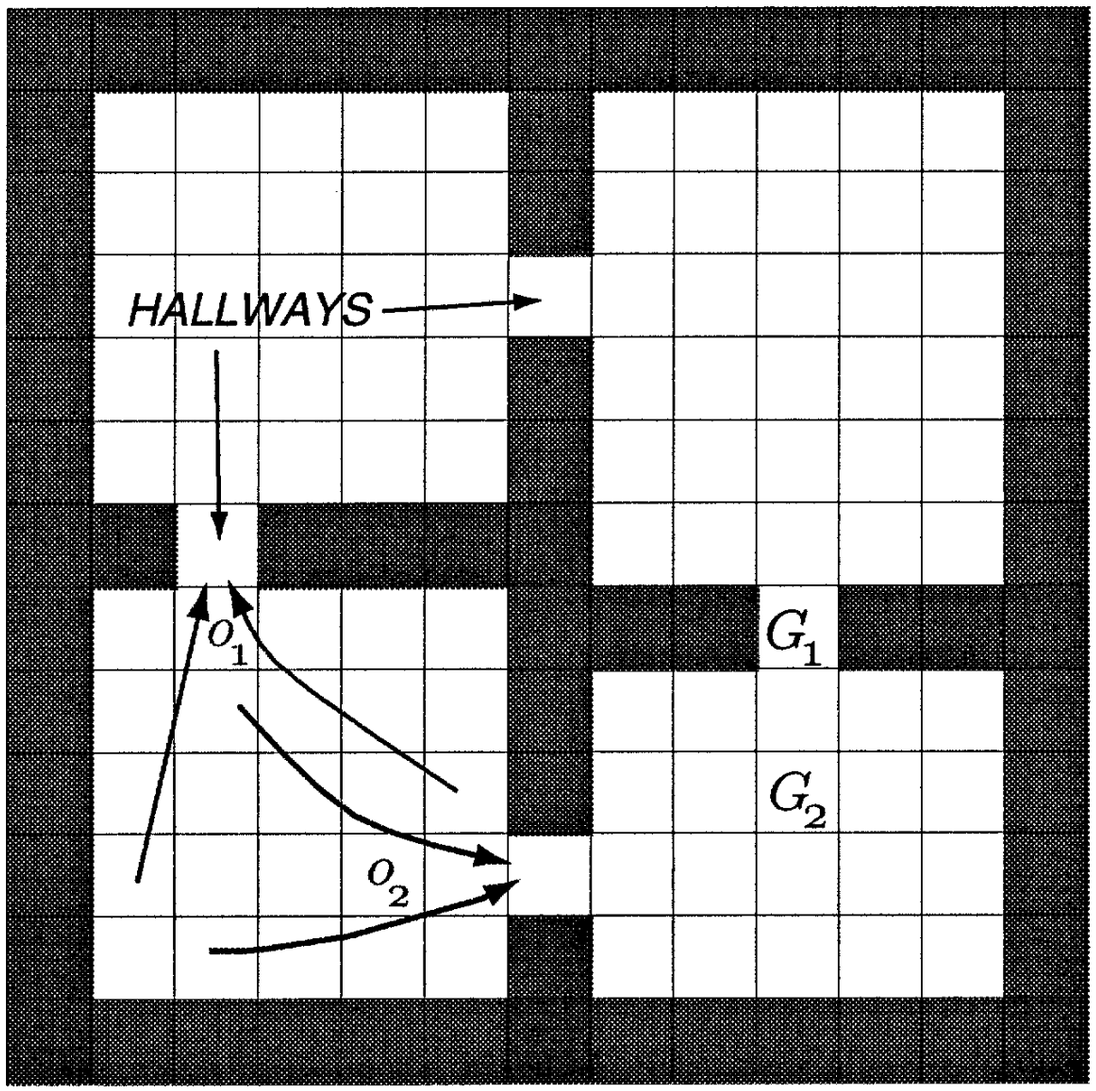
Initiation and Termination

- Initiation set $\mathcal{I}_o$
  - States from which the option may be started
  - Defines scope of states from which policy must work
- Termination conditions $\beta_o(s)$
  - Define probability of terminating for each state
  - Terminate when goal is reached or out of scope for policy

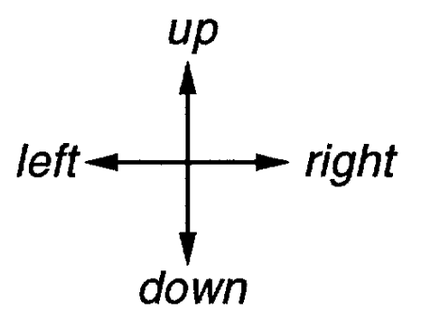


Classic Hallway Example

R.S. Sutton et al. / Artificial Intelligence 112 (1999) 181–211



4 stochastic primitive actions



Fail 33% of the time

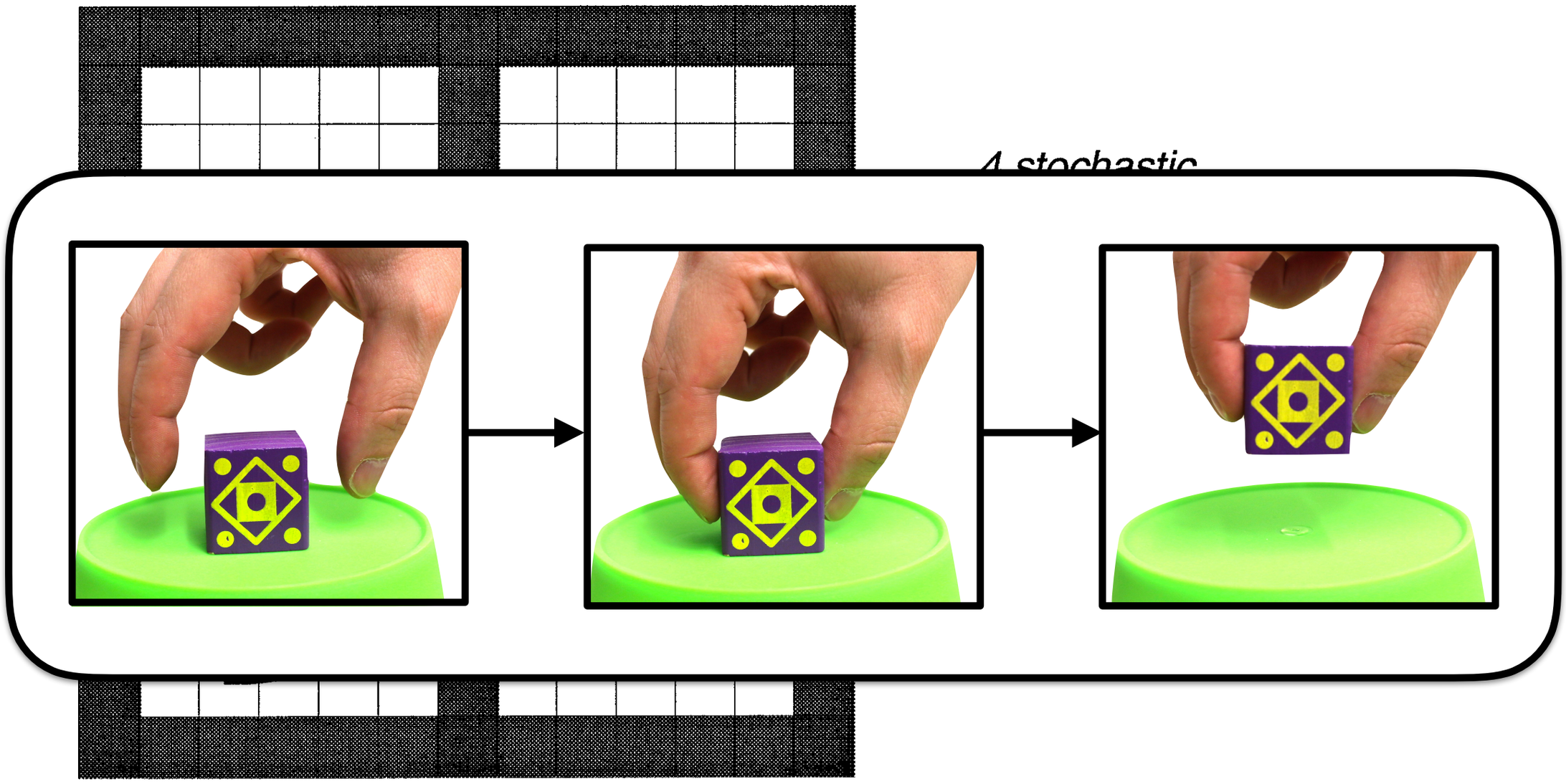
8 multi-step options

(to each room's 2 hallways)

- Learning option policy can be seen as solving smaller MDP

Classic Hallway Example

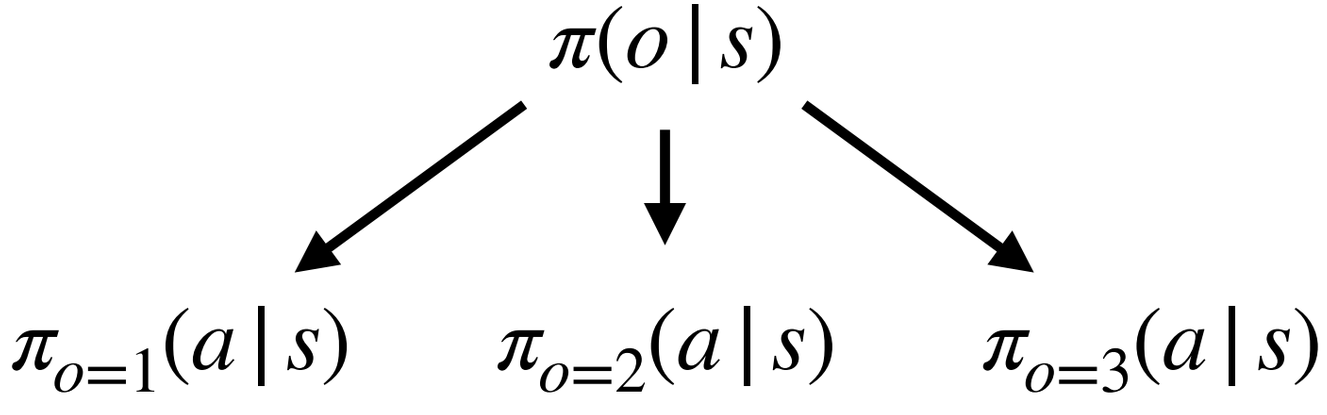
R.S. Sutton et al. / Artificial Intelligence 112 (1999) 181–211



- Learning option policy can be seen as solving smaller MDP

Hierarchical Policies

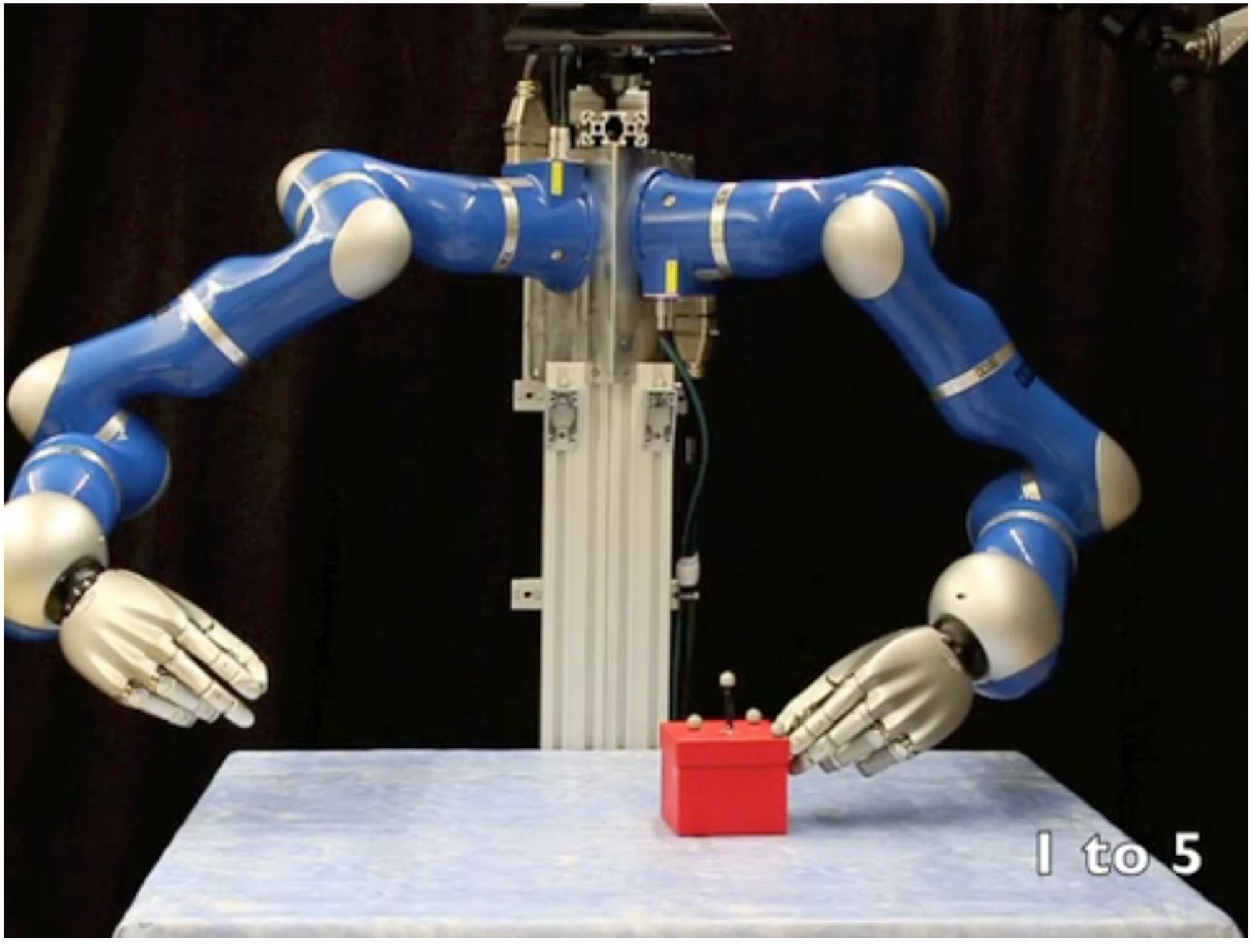
- Learn policies that select options in same way as actions



  - Take multiple steps into account for discounting rewards
  - Options can also call other options (options all the way down)
- Often call mid-level options "skills", e.g., grasp, pour, place

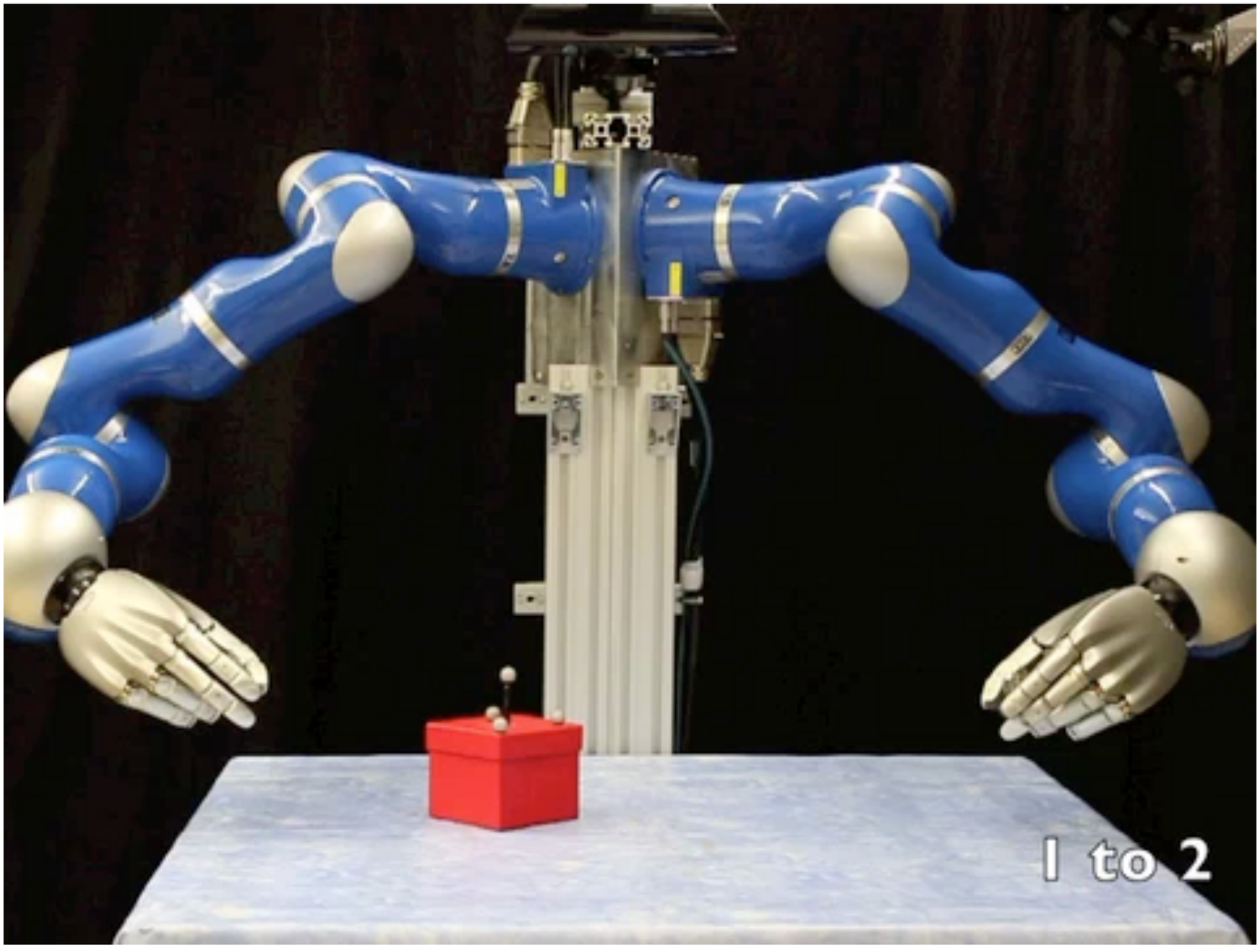
Robot Example

- Select different options based on object position



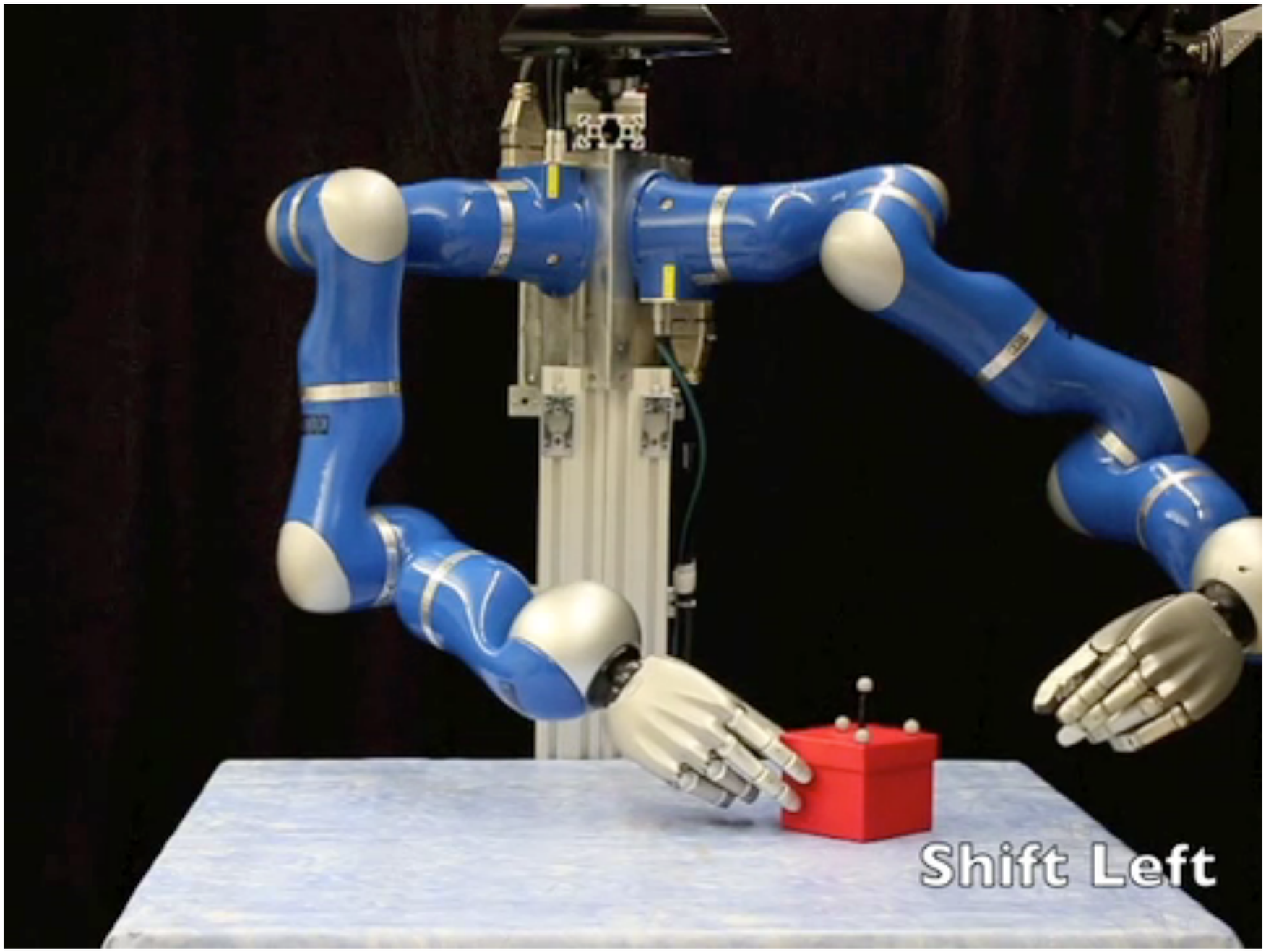
Robot Example

- Select different options based on object position



Robot Example

- Reuse skills/options between tasks



Questions?